

AD 672558



GPO PRICE \$ _____

CFSTI PRICE(S) \$ _____

Hard copy (HC) _____

Microfiche (MF) _____

ff 653 July 65

M. A. Marin

INVESTIGATION OF THE FIELD OF PROBLEMS FOR THE BOOLEAN ANALYZER

June 1968
Report No. 68-28

FACILITY FORM 602	N 68-37357	(ACCESSION NUMBER)	_____	(THRU)
	183	(PAGES)	_____	(CODE)
	CR-97264	(NASA CR OR TMX OR AD/NUMBER)	_____	08
				(CATEGORY)



DISTRIBUTION OF THIS DOCUMENT IS UNLIMITED

INVESTIGATION OF THE FIELD OF PROBLEMS FOR
THE BOOLEAN ANALYZER

Miguel A. Marin

Department of Engineering
University of California
Los Angeles, California

Distribution of this
document is unlimited.

FOREWORD

The research described in this report, "Investigation of the Field of Problems for the Boolean Analyzer," Number 68-28, by M. A. Marin, was carried out under the direction of G. Estrin, B. Bussell, W. Karplus, L. Kleinrock, D. Martin, M. Melkanoff, A. Svoboda, and L. P. McNamee, Co-Principal Investigators, in the Department of Engineering, University of California, Los Angeles.

This project is part of the continuing investigation of Variable Structure Computers, sponsored by the Atomic Energy Commission, Contract No. AT(11-1)-Gen-10, Project 14; Digital Technology, sponsored by the Office of Naval Research Contract No. Nonr 233(52); Computer Instrumentation sponsored by the Advanced Research Project Agency, DOD, Contract No. SD-184, and Large Scale Circuits sponsored by the National Aeronautics and Space Administration, Contract No. NGRO5-007-201. NGR-05-007-201

ARPA SD-184

This report was the basis of a dissertation submitted by the author.

ABSTRACT

The automation of design has been approached from many points of view. In our group here at UCLA we adopt the attitude that design automation is just another problem where a large set of statements is presented, some statements with a known, some with an unknown validity.

We want to transform problems in general to obtain a general set of Boolean equations and solve it. The number of Boolean variables in such equations is usually very large and this fact has limited the practice of approaching logical problems through the solution of Boolean equations.

Recently, a new processor, called Boolean Analyzer (BA), has been proposed by A. Svoboda. This unit operating as a part of an automatic computer is based on the idea of processing many terms of Boolean Algebra in parallel. One of its operational capabilities is the solution of large systems of Boolean equations in a reasonable time (if we compare it to standard procedures).

The Boolean Analyzer contains two basic hardware algorithms:

- 1) An algorithm for the determination of the set of prime implicants of a given Boolean function of up to 14 variables (for the first proposed model). This algorithm is based on the ordering property of the set of implicants of a Boolean function.
- 2) An algorithm for solving Boolean equations of up to 22 variables (for the first proposed model) with up to several millions of terms.

In order to promote the use of the Boolean Analyzer in its two modes of operation we developed some applications to logic problems. In this report we propose to show how the Boolean Analyzer should be integrated with a general purpose computer (or with a variable structure computer) to solve efficiently: 1) the Three-level AND-NOT synthesis problem of logic networks using only True inputs (Synthesis of TANT networks), and 2) the general problem of synthesis of logic networks using a restricted inventory

of integrated circuit modules. Our task includes reformulation of those problems into Boolean equations to prove their solubility by a Boolean Analyzer integrated with another system.

TABLE OF CONTENTS

	Page
LIST OF FIGURES	ix
LIST OF TABLES	xi
CHAPTER I – PREVIEW OF THE DISSERTATION	1
1.1 A General Survey of Methods for Solving Boolean Equations	1
1.1.1 Algebraic Methods	3
1.1.2 Matrix Methods	4
1.1.3 Tabular Methods	5
1.2 Computer Programming Solution of Systems of Boolean Equations.	16
CHAPTER II – THE UCLA VARIABLE STRUCTURE COMPUTER	19
2.1 The Fixed-Plus-Variable System	19
2.2 The Boolean Analyzer as a Part of the Variable Structure Computer	20
CHAPTER III – THE BOOLEAN ANALYZER PROPER	25
3.1 Theory	25
3.2 Components.	34
3.3 Modes of Operation.	38
3.3.1 Mode I: Non-Implicants Determination	39
3.3.2 Mode II: Non-Prime-Implicants Determination	43
CHAPTER IV – APPLICATIONS OF THE BOOLEAN ANALYZER	47
4.1 Solution of Systems of Boolean Equations	47
4.2 Irredundant Coverings of a Boolean Function.	50
4.3 Synthesis of TANT Networks	53
4.3.1 Introduction	53
4.3.2 Definitions and Theorems	54
4.3.3 Calculation of the Generalized Prime Implicants	60
4.3.4 The Selection of Generalized Prime Implicants	61
4.4 A General Synthesis Method of Logical Nets Using Fixed Inventory of Integrated Circuits	86
4.4.1 The Synthesis of Single-Output Logical Nets	86
4.4.2 Synthesis of Multiple-Output Logical Nets.	109
CHAPTER V – COMPUTER SIMULATION OF THE B.A. SYSTEM	115
5.1 Simulation of the Modes of Operation of the B.A.	115

TABLE OF CONTENTS (Continued)

	Page
5.1.1 Simulation of Mode I	117
5.1.2 Simulation of Mode II	119
5.2 Experimental Results	122
CHAPTER VI - CONCLUSIONS	127
BIBLIOGRAPHY	133
APPENDIX I - Program SOLDET	137
APPENDIX II - Program SBV3	143
APPENDIX III - Program SBEV4	149
APPENDIX IV - Boolean Analyzer Simulator (BAS)	163

LIST OF FIGURES

		Page
1.	Discriminant of a System of Equations	8
2.	Example of Discriminant	11
3.	Discriminant Decompositions	12
4.	Discriminant of a Singular System.	14
5.	Zero Columns of Discriminant	15
6.	Variable Structure System	19
7.	Block Diagram of the Boolean Analyzer	21
8.	Block Diagram and Data Flow of an Extended Version of the Boolean Analyzer	23
9.	Flowchart Corresponding to MODE I	40
10.	Flowchart Corresponding to MODE II.	44
11.	Flowchart for System of Boolean Equations.	49
12.	Flowchart for Irredundant Coverings	51
13.	Example of TANT Circuit	54
14.	TANT Synthesis Example.	59
15.	Discriminant of Equation (31).	65
16.	TANT Circuit Corresponding to Table V.	75
17.	Truth Table of Example 7	78
18.	TANT Circuit Corresponding to Example 7: First Solution	80
19.	TANT Circuit Corresponding to Example 7: Second Solution	80
20.	Flowchart of TANT Synthesis Using B.A.	85
21.	Combinational Logic Net	88
22.	Combinational Logic Net Showing Last Logic Level	89
23.	NAND Synthesis Example.	100
24.	Discriminant of Equation (38).	101
25.	WOS Synthesis Example.	108
26.	Multiple-Output Combinational Net	110
27.	Multiple-Output WOS Synthesis Example.	114

LIST OF FIGURES (Continued)

	Page
28. System of Boolean Equations Discriminant	123
29. Prime Implicants Determination	124

LIST OF TABLES

	Page
I. Space of Terms T	29
II. Cancellation of Non-Implicants	31
III. Selection of Prime Implicants Corresponding to T_C of Table II	33
IV. Modes of Operation of the B.A.	46
V. Generalized Prime Implicants	69
VI. Covering of Minterms	73
VII. T-Factors Selection	77
VIII. Selection Table	82
IX. NAND Functions of 3 Inputs	93
X. WOS Functions of 3 Inputs	104
XI. Maximum-Sharing Computation.	113
XII. Applications of the Boolean Analyzer	128

CHAPTER I

PREVIEW OF THE DISSERTATION

1.1 A general survey of methods for solving Boolean equations

The original formulation of the problem was first presented by George Boole in his "Investigation of the Laws of Thought"¹ by using variables representing the validity or non-validity of propositions which we call today Boolean variables.

To solve a system of Boolean equations we want to determine the unknown validities of certain propositions as functions of some others with known validities. We say that a set of such functions is a solution of the given system of equations when the condition defined by the set of functions implies the validity of all equations of the system. Boole proved¹ that any logical problem in general may be expressed by means of certain logical operators which represent the given problem and which can be reduced to a system of logical equations (Boolean equations).

The problems we are interested in in this dissertation are those which are related to the field of logical design of computers and we consider them in this work as an example of how problems encountered in the

design of mathematical machines may be solved using the same logic which makes possible the existence of these machines.

Although the Boolean algebra and its applications to the logical design of computers have been extensively presented in the literature, its particular application to solution of logical design problems by systems of Boolean equations attracted almost no attention. The reason for this may be that in this type of problem we get systems of Boolean equations with too large a number of Boolean variables, or that because of lack of effective methods for solving systems of Boolean equations with large number of variables we find this approach impracticable.

Today there are two effective ways in principle how to solve systems of Boolean equations: 1) by computer programming and 2) by special computer design. The computer programming methods will be presented in Section 1.2. A special computer design for solving Boolean equations, called Boolean Analyzer¹⁸, has been recently proposed by A. Svoboda and will be described in Chapters II and III.

In this section we present a panoramic view of classical methods from the literature in three main groups: Algebraic, Matrix, and Tabular (Map) methods.

1.1.1 Algebraic methods

The first algebraic method for solving systems of Boolean equations was developed by George Boole¹. This method is essentially an elimination method similar to the one used in linear algebra. The result however has to be interpreted following specific rules of interpretation of symbols like $0/0$, $1/0$. The main drawback of this method is that for equations with high number of variables the method requires a great amount of algebraic operations, the insertion of special symbols ($0/0$, $1/0$...) and a final interpretation of the solution. George Boole¹ works algebraically only with problems with small number of variables and finds the solutions for problems with high numbers of them in special cases only.

Extensive studies of algebraic methods and of the existence of solutions followed Boole's contribution. Bernstein² studied the conditions for a Boolean equation to have a unique solution. Gotō³ developed general methods for solving logic algebraic equations. Zemanek⁴, Rouche⁵, Phister⁶, Shubert⁷, Klir⁸, Carvallo⁹, Rudeanu¹⁰ and many others have done significant contributions in this field. It is, however, a common feature of the algebraic methods that these are not easily applicable to systems with a large number of variables because of the amount of algebraic operations required. Sometimes it is possible to obtain a solution by inspecting the

form of the given system. In general, however, this is not the case and therefore we do not consider it here.

Rudeanu¹⁰ proposed a general method for solving Boolean equations and presented some interesting applications to the problem of conductibilities of a multipole.

In order to computerize an algebraic method it is required to generate an algorithm that will avoid the use of algebraic expressions which are difficult to handle with computers. Therefore it will be better, from the computer implementation point of view, to transform the problem to matrix form or to use tables which in general are more suitable for computers.

1.1.2 Matrix Methods

Boolean equations represented using Boolean matrices were first used by Hohn and Schissler¹¹ in the design of combinational relay switching circuits. Campeau¹² uses Boolean matrices for the synthesis and analysis of counters in digital systems and presents a method for solving Boolean equations using matrices. This method is based on the concept of the matrix determinant of the system and is analogous to Cramer's rule in systems of linear algebraic equations. The problems that these matrix methods present for systems with a large number of variables are well known because they are similar to

those encountered in linear systems. These are: the handling of singular matrices, the matrix inversion problem, etc.

Another method which can be classified as both a matrix and a tabular method is the one developed by Ledley¹³ and extensively applied to logic circuit design¹⁴. This method is based on the concept of designation numbers which are used to represent all combinations of logical values that a given variable takes in the given system of equations. Using this concept a Boolean matrix may be associated with every Boolean equation of the system. The logical combination (bit by bit multiplication) of the matrices representing every equation gives a unique matrix which is used to determine the total number of existing solutions and the explicit algebraic form of every solution. This method is very similar to Svoboda's tabular¹⁶ method which we shall present in the next section. It should be said here that although both methods are very similar, the convenient tabular presentation of Svoboda makes easy the determination of every possible solution by means of adequate maps and also the treatment of the singular cases of certain systems (i.e. systems with no solution).

1.1.3 Tabular Methods

The use of maps for the solution of systems of

Boolean equations was first proposed by Maitra¹⁵. His map approach applies to a certain class of Boolean Functional equations only and therefore is not considered here. An extension of the map approach to the solution of general systems of Boolean equations is due to Svoboda¹⁶ who introduces for the first time the concept of logical relation between logical spaces.

He considers a logical space (a map) where the simultaneous validity of the equations of the system is plotted and where the logical space corresponding to unknown variables is separated from the logical space of the known variables. The logical dependency between these spaces gives complete information about the number and nature of the solutions including the singular cases when the known variables are not logically independent.

The statement of the problem of solving systems of Boolean equations is as follows:

Given the system of simultaneous Boolean equations

$$f_i(x_1, \dots, x_n ; y_1, \dots, y_m) = g_i(x_1, \dots, y_m) \quad (1)$$

($i = 1, 2, 3, \dots, k$), where $\{x_1, \dots, x_n\}$ and $\{y_1, \dots, y_m\}$ represent the sets of known and unknown variables respectively, it is sought all sets of functions

$$y_j = y_j(x_1, \dots, x_n) \quad (j=1, 2, \dots, m) \quad (2)$$

that satisfy the system (1).*

The logical space $\{x_1, \dots, x_n\}$ of the known variables contains 2^n points which correspond to all possible values of the variables x_1, \dots, x_n . To every point of this space an identifier x may be associated which is obtained by the formula

$$x = 2^0 x_1 + 2^1 x_2 + \dots + 2^{n-1} x_n \quad (3)$$

Similarly, to every point of the logical space $\{y_1, \dots, y_m\}$ of the unknown variables there exists an identifier y given by the formula

$$y = 2^0 y_1 + 2^1 y_2 + \dots + 2^{m-1} y_m \quad (4)$$

The logical space where the simultaneous validity of the system (1) is represented is a rectangular map containing as columns the x identifiers and as rows the y identifiers. Every entry E of this map (Figure 1)

* A set of functions of the form (2) satisfies the system (1) if the set of functions (2) implies the system (1).

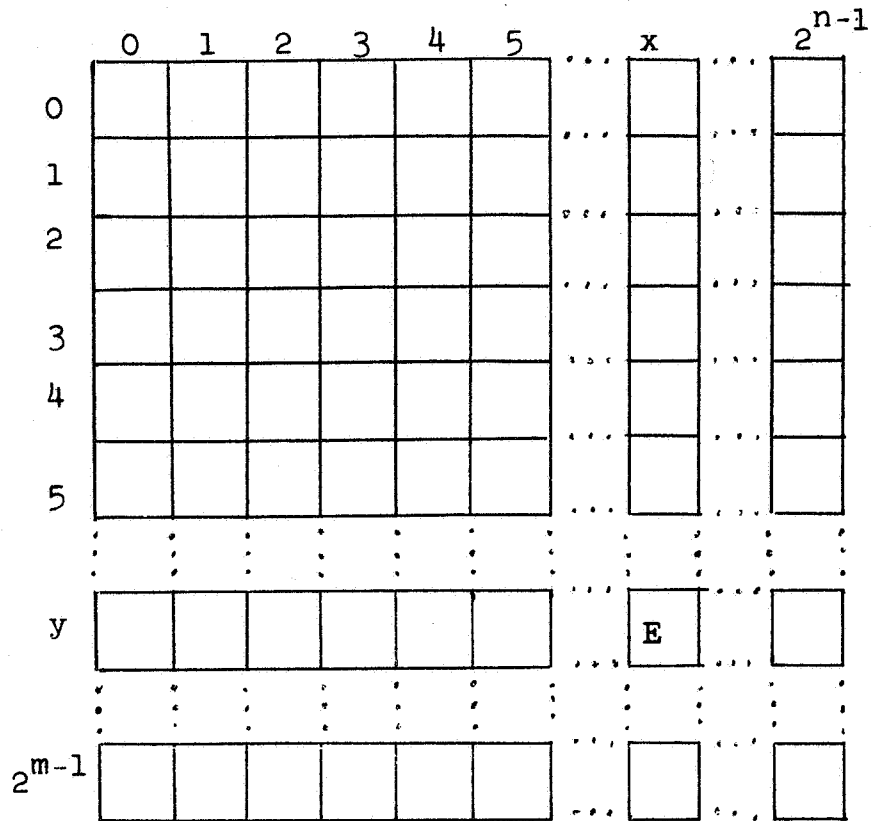


Figure 1

Discriminant of a System of Equations

is one of the 2^{n+m} possible combinations of the logical space where the system (1) is defined. The variable E will take the value 0 or 1 depending whether its corresponding combination of variables makes the system of equations (1) non-valid or valid respectively. The map thus obtained (Figure 1) is called the discriminant D of the system because it contains all information about the existing solutions (including their number).

The total number of existing solutions of the type (2) is obtained from the discriminant as follows. Let U_x represent the count of all non-zero elements in a certain column x of D , and let us form the U_x numbers for all columns x ($x = 0, \dots, 2^{n-1}$), then the total number of solutions of the form (2) is the product S of all U_x integers.

$$S = \prod_{x=0}^{x=2^{n-1}} U_x. \quad (5)$$

If $S \neq 0$ every solution (2) of (1) may be obtained by decomposition of the discriminant D into S -possible D_s ($s = 1, 2, \dots, S$) maps such that $D_s \Rightarrow D$ and every one of them contains only one non-zero element in each column.

To explain the rules of decomposition we make the following remarks:

1) Each Boolean function in the form (2) must have a unique value for every given combination of values of known variables x_1 .

2) Just a single point corresponds to that combination in the column x .

Should a decomposition map D_s of D contain more or less than one non-zero element, it would not be of the form (2).

The algebraic expressions (2) of the solutions are easily obtained from each of the D_s maps. This will be shown later in examples.

If $S = 0$, no solution of the form (2) exists and the system (1) is then said to be singular and this means that the known variables are not independent so that a definite logical relation between them must be respected to bring into existence at least one solution of the system of the form (2). Such a logical relation is obtained by equating to logical zero the logical sum of the minterms corresponding to those identifiers x of the space of known variables for which no non-zero values are found. The restricted discriminant D_s is used to obtain the solutions of the given system as in the non-singular case ($S \neq 0$), the only difference being that the rule-out columns are considered as don't care cases for possible simplification of the algebraic expression of the solutions. In the case of $S = 0$, a solution of (1) is thus an expression of the form (2) together with the equation reporting the existing logical relation between the known variables.

Examples

1) Non-singular case ($S \neq 0$)

Solve the system of Boolean equations

$$\begin{aligned}
 x_1 + \bar{x}_2 &= x_1 + y_1 \\
 y_2 + \bar{y}_1 x_1 &= x_2 y_2
 \end{aligned}
 \tag{6}$$

For convenience we first transform the system (6) into an equivalent system using the relation

$$(f = g) \iff (\bar{f}g + \bar{g}f = 0)
 \tag{7}$$

Each equation of (6) is thus transformed into the following equations respectively:

$$\bar{x}_1 x_2 y_1 + \bar{x}_1 \bar{x}_2 \bar{y}_1 = 0
 \tag{8}$$

$$\bar{x}_2 y_2 + \bar{x}_2 \bar{y}_1 + \bar{y}_2 \bar{y}_1 x_1 = 0
 \tag{9}$$

The left-hand-sides of equations (8) and (9) are now added in Boolean sense and we obtain an equation (10) which is equivalent to the given system (6).

$$\bar{x}_2 y_2 + \bar{x}_2 \bar{y}_1 + \bar{y}_2 \bar{y}_1 x_1 + \bar{x}_1 x_2 y_1 = 0
 \tag{10}$$

The discriminant D corresponding to Equation (10) is:

	0	1	2	3
0			1	
1	1	1		1
2			1	1
3				1
	1	1	2	3

Figure 2. Example of Discriminant.

The total number of existing solutions is

$$S = \prod_{x=0}^{x=3} U_x = 1 \times 1 \times 2 \times 3 = 6$$

The six decompositions of D are

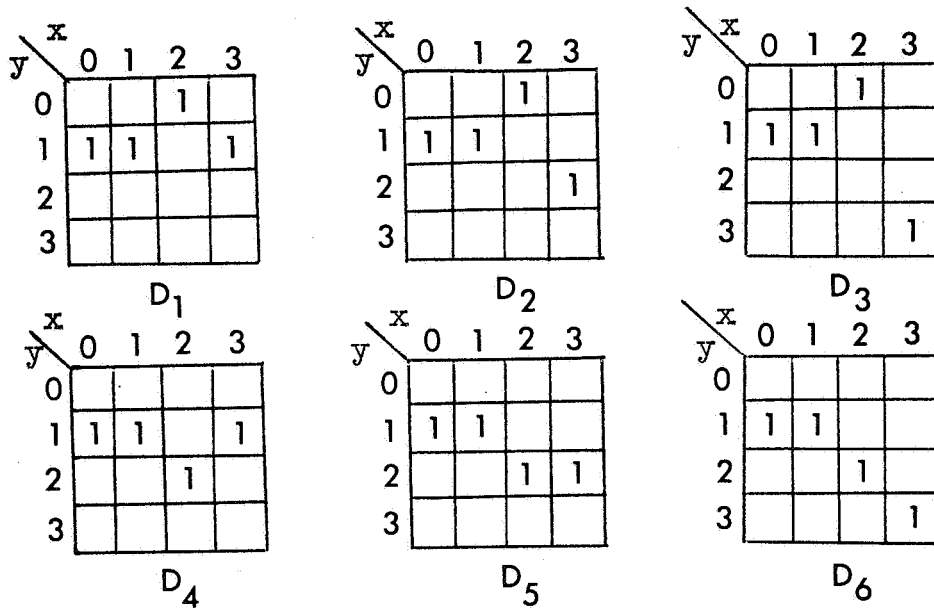
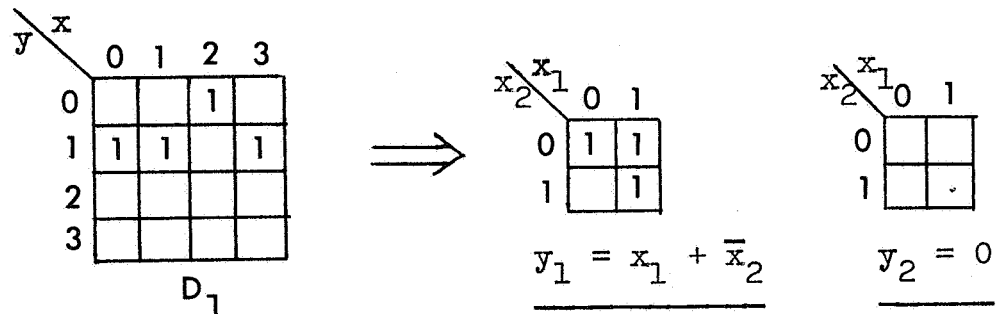


Figure 3. Discriminant Decompositions. The algebraic expression corresponding to every decomposition D_s of D is obtained using the explicit truth tables of the unknown variables as follow



	x	0	1	2	3
y	0			1	
	1	1	1		
	2				1
	3				

D_2



	x_1	0	1
x_2	0	1	1
	1		

$y_1 = x_2$

	x_1	0	1
x_2	0		
	1		1

$y_2 = x_1 x_2$

	x	0	1	2	3
y	0			1	
	1	1	1		
	2				
	3				1

D_3



	x_1	0	1
x_2	0	1	1
	1		1

$y_1 = x_1 + \bar{x}_2$

	x_1	0	1
x_2	0		
	1		1

$y_2 = x_1 x_2$

Similarly,

D_4 gives $y_1 = x_1 + \bar{x}_2$ $y_2 = \bar{x}_1 x_2$

D_5 gives $y_1 = \bar{x}_2$ $y_2 = x_2$

D_6 gives $y_1 = x_1 + \bar{x}_2$ $y_2 = x_2$

2) Singular case ($S = 0$)

System of equations

$$y_1 + y_2 = x_1 \bar{y}_2 \quad (11)$$

$$x_3 y_1 = \bar{x}_2 + y_2$$

Using the identity (7) the system (11) becomes

$$y_1 \bar{x}_1 + y_2 + x_1 \bar{y}_2 \bar{y}_1 = 0 \quad (12)$$

$$x_3 x_2 y_1 \bar{y}_2 + \bar{x}_3 \bar{x}_2 + \bar{x}_3 y_2 + \bar{y}_1 \bar{x}_2 + \bar{y}_1 y_2 = 0 \quad (13)$$

The left-hand-sides of Equations (12) and (13) may be added together reducing the original system (11) to the following equation:

$$y_1 \bar{x}_1 + y_2 + x_1 \bar{y}_2 \bar{y}_1 + x_3 x_2 y_1 \bar{y}_2 + \bar{x}_3 \bar{x}_2 + \bar{y}_1 \bar{x}_2 = 0 \quad (14)$$

The discriminant D is represented in Figure 4.

		x							
y		0	1	2	3	4	5	6	7
	0			1				1	
	1				1		1		
	2								
	3								
U	→	0	0	1	1	0	1	1	0
	x								

Figure 4
Discriminant of a Singular System

The number of solutions are

$$S = \prod_{x=0}^{x=7} U_x = 0 \times 0 \times 1 \times 1 \times 0 \times 1 \times 1 \times 0 = 0$$

The logical relation between known variables is obtained considering the columns of D with no non-zero elements (Figure 5).

y \ x	0	1	2	3	4	5	6	7
0			1				1	
1				1		1		
2								
3								

Figure 5. Zero Columns of Discriminant

This logical relation is

$$\bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 + x_1 x_2 x_3 = 0 \quad (15)$$

If we rule-out from D the columns with all-zero elements we obtain the reduced discriminant D_y which gives in this case a unique decomposition and hence only one solution:

x3 \ x2 x1	00	01	10	11
0				1
1		1		

$$\underline{y_1 = x_1}$$

x3 \ x2 x1	00	01	10	11
0				
1				

$$\underline{y_2 = 0}$$

The solution of the system of Equation (11) is

$$y_1 = x_1 \quad (16)$$

$$y_2 = 0 \quad (17)$$

$$\bar{x}_2 \bar{x}_3 + \bar{x}_1 \bar{x}_2 + x_1 x_2 x_3 = 0 \quad (15)$$

Effectively, if we substitute (16) and (17) in the original system we obtain

$$x_1 = x_1 \quad \text{which is an identity}$$

$$\text{and } x_3 x_1 = \bar{x}_2.$$

Using identity (7) it can be shown that this last equation is equivalent to

$$x_1 x_3 x_2 + (\bar{x}_1 + \bar{x}_3) \bar{x}_2 = 0$$

which is identical to relation (15). Therefore equations (16), (17) and (15) form the solution of the given system.

1.2 Computer Programming Solution of Systems of Boolean Equations.

The Svoboda's tabular method described in Section 1.1.3, can be applied to solve systems with any number of Boolean variables. When this number is larger the hand method is not practicable and a computer must be used.

When too large, the computing time of standard computers becomes prohibitively large and special operation hardware (proposed by Svoboda in 18) is necessary.

A FORTRAN IV program has been developed¹⁷ (Appendix I) which solved systems with up to 9 variables*. This program accepts equations of the type (6) computes and prints out the discriminant of the system, prints out the total number of existing solutions and the truth table of every solution.

In the singular cases, a printout of the logical relation between known variables is given.

Any of the standard general purpose digital machines available today executes a computer program in a sequential fashion, i.e. instruction after instruction. This means that any computer program that will implement any classical algorithm (Section 1.1.3) for solving Boolean equations has to process each Boolean term of the system separately (in a sequential manner). This fact imposes a basic limitation on any software implementation, of any method for solving Boolean equations.

* In principle, programs to solve systems with higher numbers of variables can be developed. However, the recently proposed Boolean Analyzer¹⁸ by far dissuades any programming effort as we shall see later.

It is possible, however, to achieve parallel processing in Boolean Algebra by using special hardware equipment¹⁸ as a part of a digital computing system. This parallel processing of Boolean terms, first introduced by Svoboda¹⁸, makes possible to compute the discriminant of the given system of Boolean equations in a very short time compared with the time required in a sequential software-type implementation. For that reason no further effort was made in developing optimal computer programs for solving Boolean equations.

In the following chapters we shall consider in detail how parallel processing of terms in Boolean Algebra can be achieved and how it applies to the solution of systems of Boolean equations and to problems stated in terms of such systems.

CHAPTER II

THE UCLA VARIABLE STRUCTURE COMPUTER SYSTEM

2.1 The Fixed-Plus-Variable System

In 1960 Estrin¹⁹ proposed a new concept in computer organization called a variable structure computer system. This system combines the characteristics of general-purpose and special-purpose computers to achieve effective computer solution of a vast class of problems.

The system, also called Fixed-Plus-Variable system, consists of a general-purpose Fixed structure computer (F) and a variety of high-speed special purpose problem oriented subsystems which form its Variable structure part (V). A control unit called Supervisory Control (SC) coordinates both structures (Figure 6).

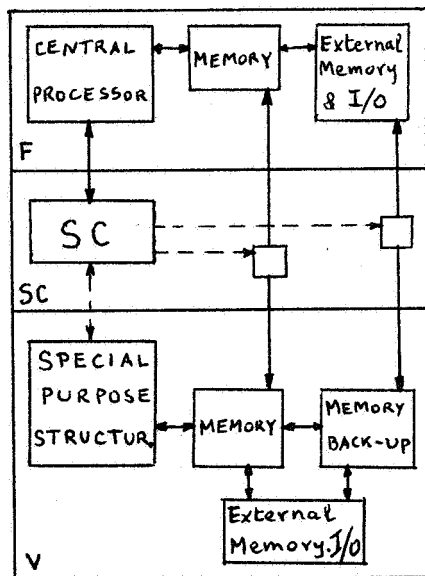


Figure 6
Variable Structure System

2.2 The Boolean Analyzer as a part of the UCLA Variable Structure Computer

The organization of the variable structure computer offers the possibility of solving systems of Boolean equations in a more efficient way than using a software implementation of any classical algorithm. A hardware V-structure implementing the algorithm may be addressed and monitored by the general purpose computer. The efficiency of such an approach is evidently higher than the all-software implementation, however, it does not offer any parallel processing feature within the V-structure. Svoboda's new theoretical results on the ordering of implicants of a Boolean Function²⁰ made possible the development of another new algorithm¹⁸ which leads to the design of a special hardware operation unit called Boolean Analyzer¹⁸.

The Boolean Analyzer system as proposed in¹⁸ (Figure 7) is composed of a general purpose computer (SDS, Figure of computer) a Circulating Memory and the Boolean Analyzer unit proper which includes a logical operational unit, a control unit, a snake delay unit and proper interface units. A more detailed description of the system is given in Section 3.2.

For the purpose of this chapter it is sufficient to say that the logical operation unit of the Boolean

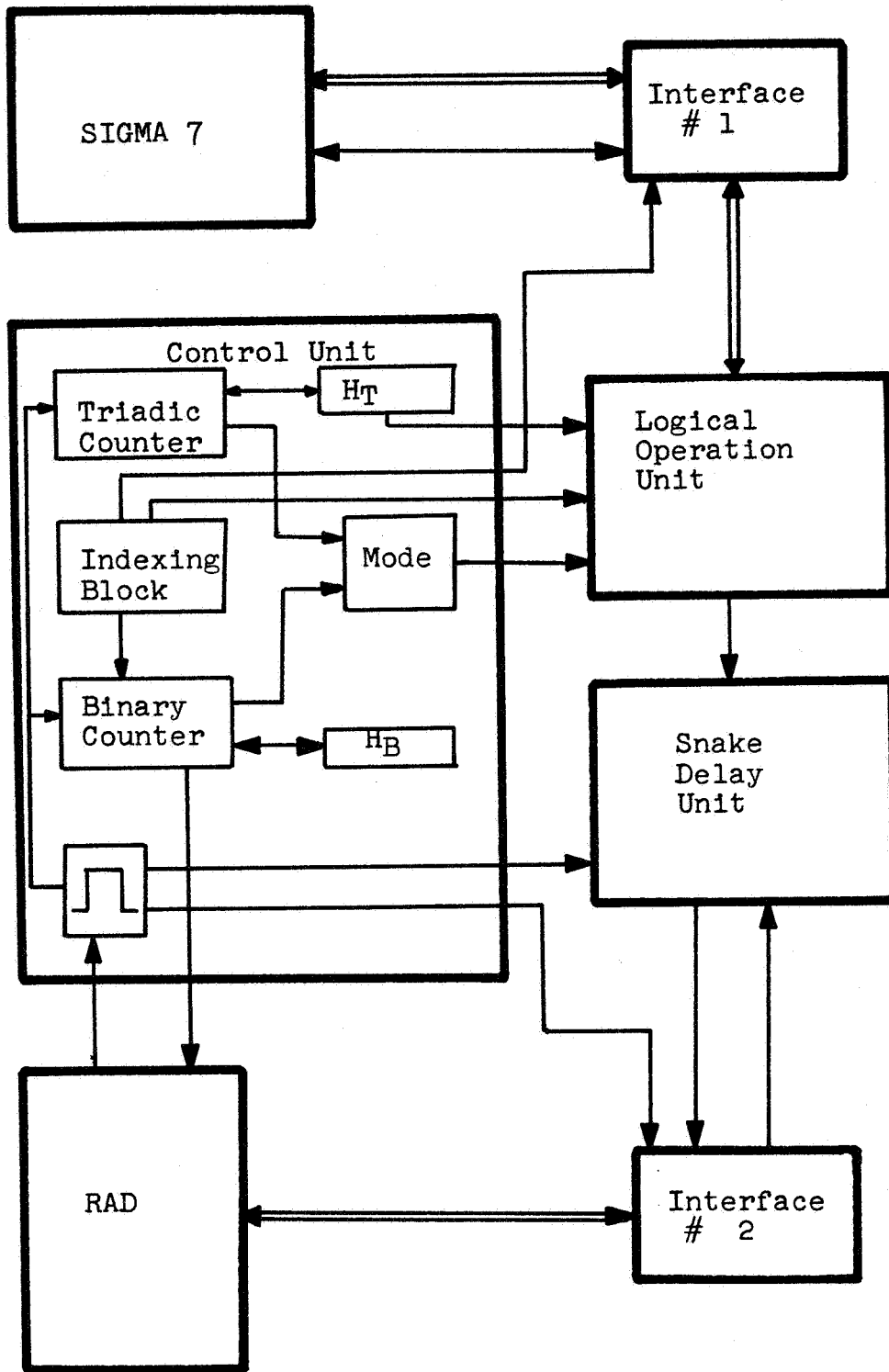


Figure 7. Block Diagram of the Boolean Analyzer ²¹

Analyzer as it stands today²¹ performs the parallel processing of the Boolean terms. To explain it, let us have a system of Boolean equations written in the form $F = 0$ (as in Equation (10)) and its terms stored in parallel in special registers of the logical operation unit. A triadic counter counts sequentially from 0 to 3^{n+m} (n and m represents the number of known and unknown variables respectively) and its content is compared in parallel with every Boolean term in the special registers. The result of each such comparison is a bit of information which is registered in the circulating memory.

It is clear that for an equation of $n+m$ total number of variables, 3^{n+m} sequential comparisons are required. This number is independent of the number of Boolean terms contained in the special registers.

Estrin's concept of variable structure computer organization permits reduction in the 3^{n+m} required sequential comparisons by using instead of a unique triadic counter any power of 3 of them ($N = 3^?$) working in parallel (Figure 8). Each triadic counter, TC_i , ($i = 1, 2, \dots, N$) should count from

$[(i-1) \cdot p]$ to $[i \cdot p - 1]$ where $i = 1, 2, \dots, N$ and $p = \frac{3^{n+m}}{N}$.

The total processing time of the logic operation unit is thus reduced by the factor of N .

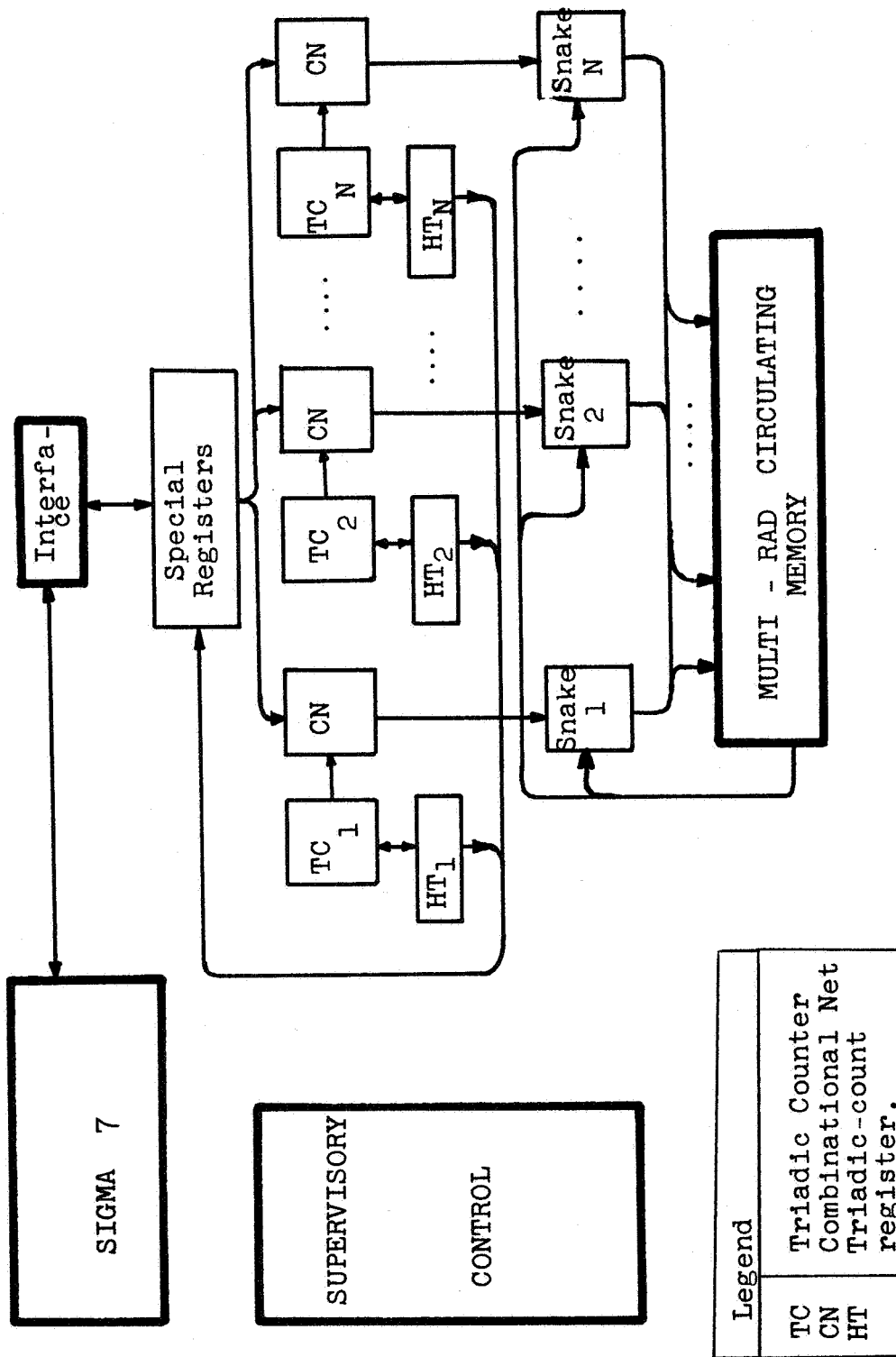


Figure 8. Block Diagram and Data Flow of an extended Version of the B.A.

The circulating memory should now provide data storage for N simultaneous data channels (Multi-RAD circulating memory, Figure 8).

Finally, a supervisory control is required in addition to the original control unit.

CHAPTER III
THE BOOLEAN ANALYZER PROPER

3.1 Theory

The Boolean Analyzer as proposed in reference 18 has two basic modes of operation: 1) Computation of the discriminant of a system of Boolean equations and 2) Prime implicants determination of a given Boolean function. The theoretical basis for the Boolean Analyzer is extensively presented by Svoboda in reference 18. In essence there are two fundamental theorems that make the parallel processing of Boolean terms possible. The first deals with the determination of non implicant min-terms of a Boolean function, the second with their ordering.

In this section we present these two theorems with detailed examples of their application. Their proof may be found in reference 18.

Let $Y = \sum_h t_h = 0$ be a $\Sigma\Pi$ - form corresponding to the complement of the given Boolean function y .

$$Y = t_a + t_b + \dots + t_h + \dots = \sum_h t_h. \quad (18)$$

Each term t_h is a Boolean function of the form

$$t_h = \bar{x}_n \bar{x}_{n-1} \dots \bar{x}_2 \bar{x}_1. \quad (19)$$

where the variable \ddot{x}_i takes on one of the values 1, x_i or \bar{x}_i exclusively, (for every $i = 1, \dots, n$, $n =$ number of variables of y). It is clear that there are 3^n possible Boolean functions of the form (19) which may be considered as elements of a 3^n -logical space T . The subscript h is the identifier of the elements t_h of the space T .

The value of h in (20) ranges from 0 to $3^n - 1$.

$$h = h_1 3^0 + h_2 3^1 + \dots + h_n 3^{n-1} = \sum_{i=1}^n h_i 3^{i-1}. \quad (20)$$

The correspondence (21) between \ddot{x}_i and h_i being established

h_i	0	1	2	(21)
\ddot{x}_i	1	x_i	\bar{x}_i	

the identifier h in (20) of every term (19) is uniquely determined and a one-to-one correspondence exists between every integer in the set $\{h\}$ and every Boolean term (19).

Example: Let $\gamma = \bar{x}_4 x_2 x_1 + x_2 \bar{x}_1 + x_4 x_3 \bar{x}_2 x_1 = t_a + t_b + t_c = 0$ and $n = 4$.

Using (21) we obtain

	x_4	x_3	x_2	x_1	3^3	3^2	3^1	3^0	
					h_4	h_3	h_2	h_1	h
t_a	x_4		x_2	x_1	2	0	1	1	58 = a
t_b			x_2	\bar{x}_1	0	0	1	2	5 = b
t_c	x_4	x_3	\bar{x}_2	x_1	1	1	2	1	43 = c

$$Y = t_{58} + t_5 + t_{43} .$$

Each term t_h in Equation (18) implies Y , but $y = \bar{Y}$ by definition of Y , therefore the terms t_h are nonimplicants of y . In other words every term t_h implicant of Y is a non-implicant of y .

In order to determine all implicants of y it is sufficient to cancel from the 3^n space T of all possible terms those which are non-implicants of y .

This statement is proven by the following theorem.

Theorem 1. (Svoboda (18))

Given a Boolean function y and its complement Y , let $\{t_h\}$ be the set of implicants of Y , and $\{t_{h'}\}$ represent all terms in T .

The sufficient condition for a definite $t_{h'}$, ($h' = \sum_{j=1}^n h'_j \cdot 3^{j-1}$, $t_{h'} \in \{t_{h'}\}$), to be a non-implicants of y is that for at least one $t_{h'}$ - term of Y , ($h = \sum_{j=1}^n h_j \cdot 3^{j-1}$, $t_h \in \{t_h\}$),

$$h_j + h'_j \neq 3 \text{ for } j = 1, 2, 3, \dots, n \quad (22)$$

Example 1.

Let $y = x_2 + \bar{x}_3 x_1$ ($n = 3$). The complement function Y is

$$Y = \bar{y} = \bar{x}_2(x_3 + \bar{x}_1) = \bar{x}_2 x_3 + \bar{x}_2 \bar{x}_1$$

The set $\{t_h\}$ has two terms: $t_{15} = x_3 \bar{x}_2$

$$t_8 = \bar{x}_2 \bar{x}_1$$

The triadic representation of their identifiers are

$t_{15} : (2 \ 1 \ 0)$, $t_8 : (0 \ 2 \ 2)$. (Table I)

The space T is composed of $3^3 = 27$ $t_{h'}$ -terms which is represented in Table I (see following page).

TABLE I
SPACE OF TERMS T

{ t_h }			T-space			
t_h	h	$h_3 \ h_2 \ h_1$	$h'_3 \ h'_2 \ h'_1$	h'	$t_{h'}$	
$\bar{x}_2 \bar{x}_1$	8	0 2 2	0 0 0	0		
$x_3 \bar{x}_2$	15	1 2 0	0 0 1	1	x_1	
			0 0 2	2	\bar{x}_1	
			0 1 0	3	x_2	
			0 1 1	4	$x_2 x_1$	
			0 1 2	5	$x_2 \bar{x}_1$	
			0 2 0	6	\bar{x}_2	
			0 2 1	7	$\bar{x}_2 x_1$	
			0 2 2	8	$\bar{x}_2 \bar{x}_1$	
			1 0 0	9	x_3	
			1 0 1	10	$x_3 \ x_1$	
			1 0 2	11	$x_3 \ \bar{x}_1$	
			1 1 0	12	$x_3 x_2$	
			1 1 1	13	$x_3 x_2 x_1$	
			1 1 2	14	$x_3 x_2 \bar{x}_1$	
			1 2 0	15	$x_3 \bar{x}_2$	
			1 2 1	16	$x_3 \bar{x}_2 x_1$	
			1 2 2	17	$x_3 \bar{x}_2 \bar{x}_1$	
			2 0 0	18	\bar{x}_3	
			2 0 1	19	$\bar{x}_3 \ x_1$	
			2 0 2	20	$\bar{x}_3 \ \bar{x}_1$	
			2 1 0	21	$\bar{x}_3 x_2$	
			2 1 1	22	$\bar{x}_3 x_2 x_1$	
			2 1 2	23	$\bar{x}_3 x_2 \bar{x}_1$	
			2 2 0	24	$\bar{x}_3 \bar{x}_2$	
			2 2 1	25	$\bar{x}_3 \bar{x}_2 x_1$	
			2 2 2	26	$\bar{x}_3 \bar{x}_2 \bar{x}_1$	

Table II displays the step-by-step application of condition (22). The implicants of y form the set T_c . This set T_c has an ordering property²⁰ that leads to the selection of prime implicants of y .

Definition: Definition of prime-implicant.

Let t_h be an implicant of y , $t_h \in T_c$, and let $t_{h'}$ be an implicate of t_h , $t_h \Rightarrow t_{h'}$; We say that t_h is a prime-implicant of y if the following proposition

$$[(t_h \Rightarrow y) \wedge (t_h \Rightarrow t_{h'}) \wedge (t_h \neq t_{h'})] \Rightarrow (t_h \not\Rightarrow y) \quad (24)$$

is true for all $t_{h'}$.

Theorem 2: Ordering of implicants. (Svoboda²⁰)

If a term $t_a \Rightarrow t_b$, then $a \geq b$. (25)

Theorem 3: Exclusion of non-prime-implicants.

If a term t_h , $t_h \in T_c$, is a prime implicant of y then any other term t_k implicant of t_h is not a prime implicant of y .

Example 2: Ordering of implicants (see Table II).

$$t_4 = x_2 x_1, t_3 = x_2, [(t_4 \Rightarrow t_3) \Rightarrow (4 > 3)].$$

TABLE II
 CANCELLATION OF NON-IMPLICANTS

Implicants of y	T_c	$h_3' h_2' h_1'$	h_j+h_j'	h_j+h_j'	h_j+h_j'	$\{t_h\}$	T	h'
	CANCELLED	0	0	0	2	0	0	0
	CANCELLED	1	1	2	3	0	1	1
	CANCELLED	2	2	2	4	2	2	2
x_2	0	0	1	3	2	1	0	3
$x_2 x_1$	0	1	1	3	3	1	1	4
$x_2 x_1$	0	1	2	3	3	1	2	4
	CANCELLED	1	4	4	4	2	2	5
	CANCELLED	2	4	4	4	2	2	6
	CANCELLED	3	4	4	4	2	2	7
	CANCELLED	4	2	2	2	0	0	8
	CANCELLED	5	2	2	2	0	0	9
	CANCELLED	6	2	2	2	0	0	10
	CANCELLED	7	2	2	2	0	0	11
	CANCELLED	8	2	2	2	0	0	12
	CANCELLED	9	2	2	2	0	0	13
$x_3 x_2$	1	1	0	1	2	1	1	14
$x_3 x_2 x_1$	1	1	1	2	3	1	1	15
$x_3 x_2 x_1$	1	1	2	3	3	1	1	16
	CANCELLED	2	4	4	4	2	2	17
	CANCELLED	3	4	4	4	2	2	18
	CANCELLED	4	2	2	2	0	0	19
$\bar{x}_3 x x_1$	2	0	1	3	2	1	2	20
	CANCELLED	2	2	2	2	0	0	21
$\bar{x}_3 x_2$	2	1	0	3	2	1	2	22
$\bar{x}_3 x_2 x_1$	2	1	1	3	3	1	2	23
$\bar{x}_3 x_2 x_1$	2	1	2	3	3	1	2	24
	CANCELLED	2	4	4	4	2	2	25
$\bar{x}_3 \bar{x}_2 x_1$	2	2	1	3	2	1	2	26
	CANCELLED	2	2	2	2	0	0	

Example 3: Exclusion of non-prime-implicants

(see Table III).

The term with the lowest identifiers $t_3 \equiv x_2$ is a prime implicant because it has only one implicate, namely itself $t_{h'} = x_2$, which is ruled out by relation (24). In consequence and by virtue of theorem 3, terms $t_4, t_5, t_{12}, t_{13}, t_{14}, t_{21}, t_{22}, t_{23}$, are cancelled in T_c as non-prime-implicants of y . Term $t_{19} \equiv \bar{x}_3 x_1$ is a prime implicant because its implicates (ruling out $\bar{x}_3 x_1$) are $t_{18} \equiv \bar{x}_3$ and $t_1 \equiv x_1$ which are CANCELLED in T_c meaning that they are non-implicants of y .

Based on the above concepts and theorems, the algorithm for the prime-implicants selection of y has the following three steps:

- 1) Determination of the ordered set T_c containing all implicants of y .
- 2) The term $t_p, t_p \in T_c$, with smaller identified p is a prime-implicant of y and is transferred to the set $\{t_h\}$. ($\{t_h\}$ is empty at the start of the algorithm and contains all prime implicants of y at the end)
- 3) All implicants of t_p are cancelled in T_c .

Steps 2 and 3 are repeated until T_c is empty.

TABLE III
 SELECTION OF PRIME IMPLICANTS
 CORRESPONDING TO T_c OF TABLE II

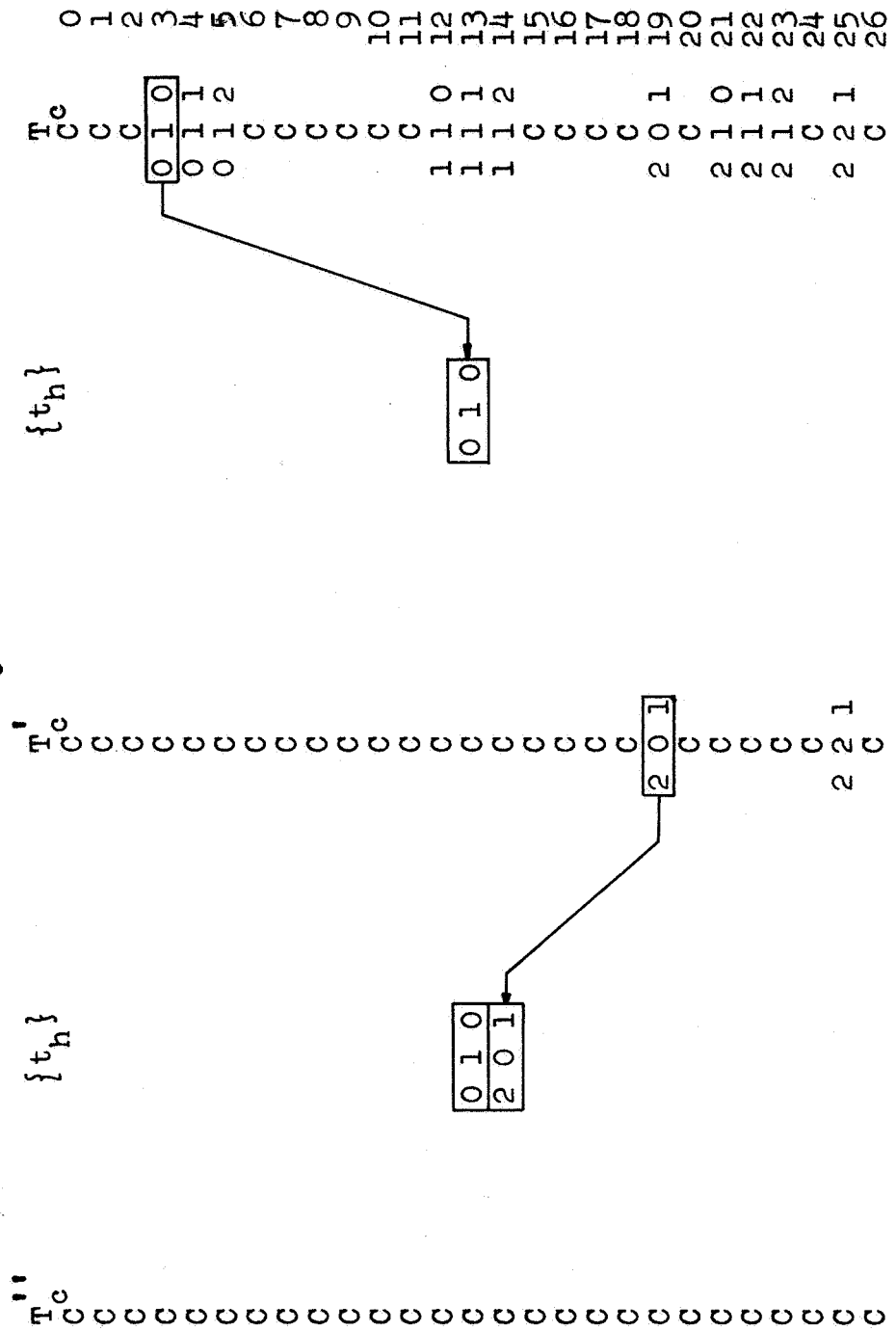


Table III displays the application of Steps 2 and 3 to example 1.

3.2 Components

The logical design of the Boolean Analyzer (BA) is presented in detail in reference 21. In this section, and for the purpose of its simulation (Chapter V), we describe briefly the hardware components and characteristics of the Boolean Analyzer.

The Boolean Analyzer System (Figure 7) is composed of three subsystems:

- 1) A general purpose computer, (SDS, SIGMA 7)
- 2) A circulating memory : for example, the SDS Rapid - Access-Data (RAD) storage system.
- 3) The Boolean Analyzer Unit (BAU) proper.

BAU is composed of the following parts:

- 1) An Interface #1 between the general purpose computer and the BA.
- 2) An Interface #2 between the BAU and the circulating memory.
- 3) The logic operation unit (LOU)
- 4) The control unit
- 5) The snake unit

3.2.1 The general purpose computer is a Scientific Data Systems SIGMA 7 Computer, whose READ DIRECT and WRITE DIRECT instructions facilitate its use in real-time, time-sharing, and multiusage applications.

3.2.2 The Circulating memory (SDS RAD SYSTEM, Models 7203/7201) includes 256 tracks, 16 sectors/track, 360 bytes/track at a rate of 5μ seconds/8 bit-byte and contains only one read/write amplifier with proper electronic switching. Its data may be organized in two modes:

Mode A: 27 groups/sector, each group with 88 bits

Mode B: 64 groups/sector, each group with 40 bits.

3.2.3 The logical design of Interfact #2 between the BAU and the circulating is presented in detail in reference 22 and is composed of an 8-bit read-shift-register (Reading Buffer) and an 8-bit write-shift-register (Writing Buffer).

3.2.4 LOU contains special registers for storing triadic numbers (the identifiers h of the terms in $Y = 0$) and special combinatorial network which generates a signal C_k according to its switching mode:

Mode A: $(h_j^k + h'_j \neq \text{for every } j) \Rightarrow (C_k = 1)$

(Theorem 1, Section 3.1)

$$\text{Mode B: } \{(h_j^k + h'_j \neq 3) \wedge [(h_j^k = 0) \\ \Rightarrow (h'_j = 0)]\} \Rightarrow (C_k = 1)$$

(Theorem 3, Section 3.1).

3.2.5 The Interface #1 provides the linkage between the BAU and SIGMA 7 computer and contains proper encoding and decoding devices to store and retrieve data from the registers of the LOU.

3.2.6 The Control Unit includes: 1) a binary counter, counting from 0 to $2^{22}-1$; 2) a triadic counter, counting from 0 to $3^{14}-1$; 3) an Indexing Block whose mission is to provide proper indexing of flip-flops in the LOU; 4) a pulse generator; 5) two special registers, H_t and H_b for temporary storage of the contents of the triadic and binary counters respectively.

3.2.7 The Snake Delay Unit logically combines the information coming from two different sources, the RAD and the LOU, whose organization has different bases: the RAD is organized in 8-bit bytes while the LOU may have a triadic organization. The Snake Delay Unit contains two delay logical circuits, Snake #1 and Snake #2, with proper control for two-way information transfer between

RAD and LOU. Two modes of data organization are possible: Mode a which transforms a 9-bit input sequence into an 8-bit output sequence and Mode b which transforms an 8-bit input sequence into a 9-bit output sequence.

Svoboda¹⁸ gives operation time estimates for the Boolean Analyzer. Its operation time is a function of the number of variables and the size of the LOU special registers. For a model with 100 special registers the following estimates were given¹⁸:

1) For discriminant computation the time ranges from .005 sec for 500 terms of 10 variables to 2 hours for 500,000 terms of 20 variables.

2) For prime implicants determination the time ranges from .0022 sec for 100 terms of 7 variables to 383 sec for 8,000 terms of 14 variables.

The operation time increases logarithmically (as a rough approximation) with the number of variables.

The circulating memory is mainly the device that imposes a limitation on the number of variables. The SDS RAD system, for example, has a capacity of 12 million bits. For functions with 15 variables, $3^{15} \approx 14$ million bits are required in the circulating memory which exceed the RAD storage capacity. The utilization of a multi-RAD memory system (Figure 8) could overcome this limitation.

A great advantage of the parallel structure of the LOU is that it can easily be extended to allocate higher numbers of variables, because the same simple circuit is used for each variable.

3.3 Modes of Operation

The Boolean Analyzer unit operates in two basic modes: Mode I corresponding to the realization of Theorem 1 (Section 3.1) and Mode II corresponding to the implementation of Theorem 3 (Section 3.1).

These two modes of operation constitute the elementary logic operations of the Boolean Analyzer and consequently any given problem to be solved with this system must be formulated in terms of them. This formulation as it applies to certain problems shall be the task of Chapter IV. In this section we present the procedures* corresponding to Modes I and II as they are executed by the Boolean Analyzer Unit (Figure 7).

*More details on these procedures may be found in the works of Lonnie Laster¹⁸ and George Gilley²².

3.3.1 MODE I: Non-implicants determination

This mode of operation corresponds to the determination of non-implicants of a Boolean function.

If all non-implicant terms are desired, then the B.A. operates in triadic mode, using the Triadic Counter; the mode selection is made in the mode control.

If only non-implicant minterms are sought, then the mode control selects the binary counter and the B.A. operates in binary mode.

The procedure corresponding to MODE I is outlined in the flowchart of Figure 9. The following initial conditions which can either be set manually or automatically by the digital computer (SIGMA 7), are required:

- 1) The mode control selects the triadic counter (TC) and the binary counter (BC)
- 2) LOU registers are organized either in Mode a (corresponding to mode control selection of TC) or Mode b when BC is selected. LOU combinatorial network is connected in Mode A.
- 3) Snake Unit is connected either in Mode A (corresponding to selection of TC) or Mode B, corresponding to selection of (BC)
- 4) Circulating Memory, either in Mode A (TC selection) or Mode B (BC selection), is filled

Figure 9. Flowchart corresponding to MODE I.

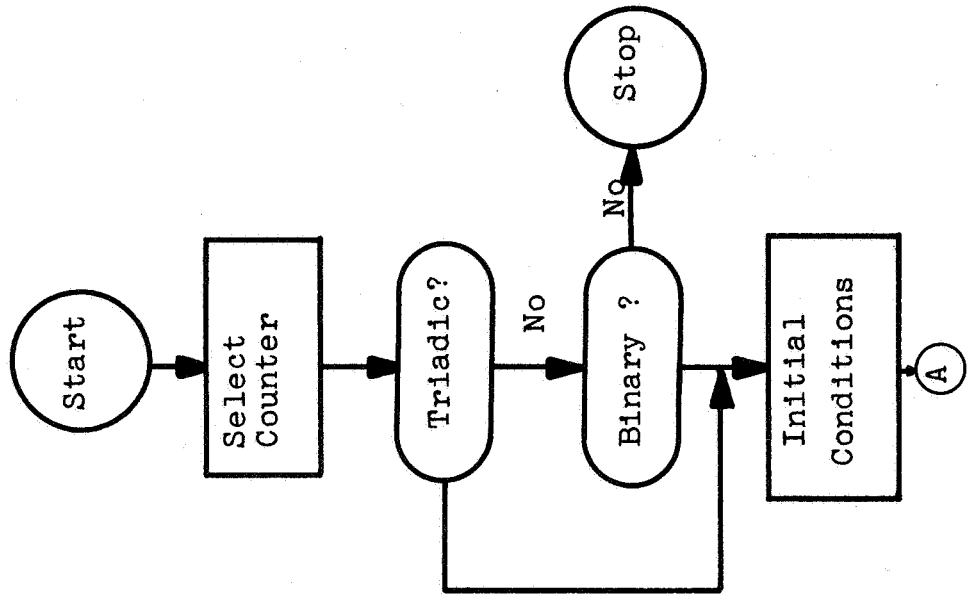
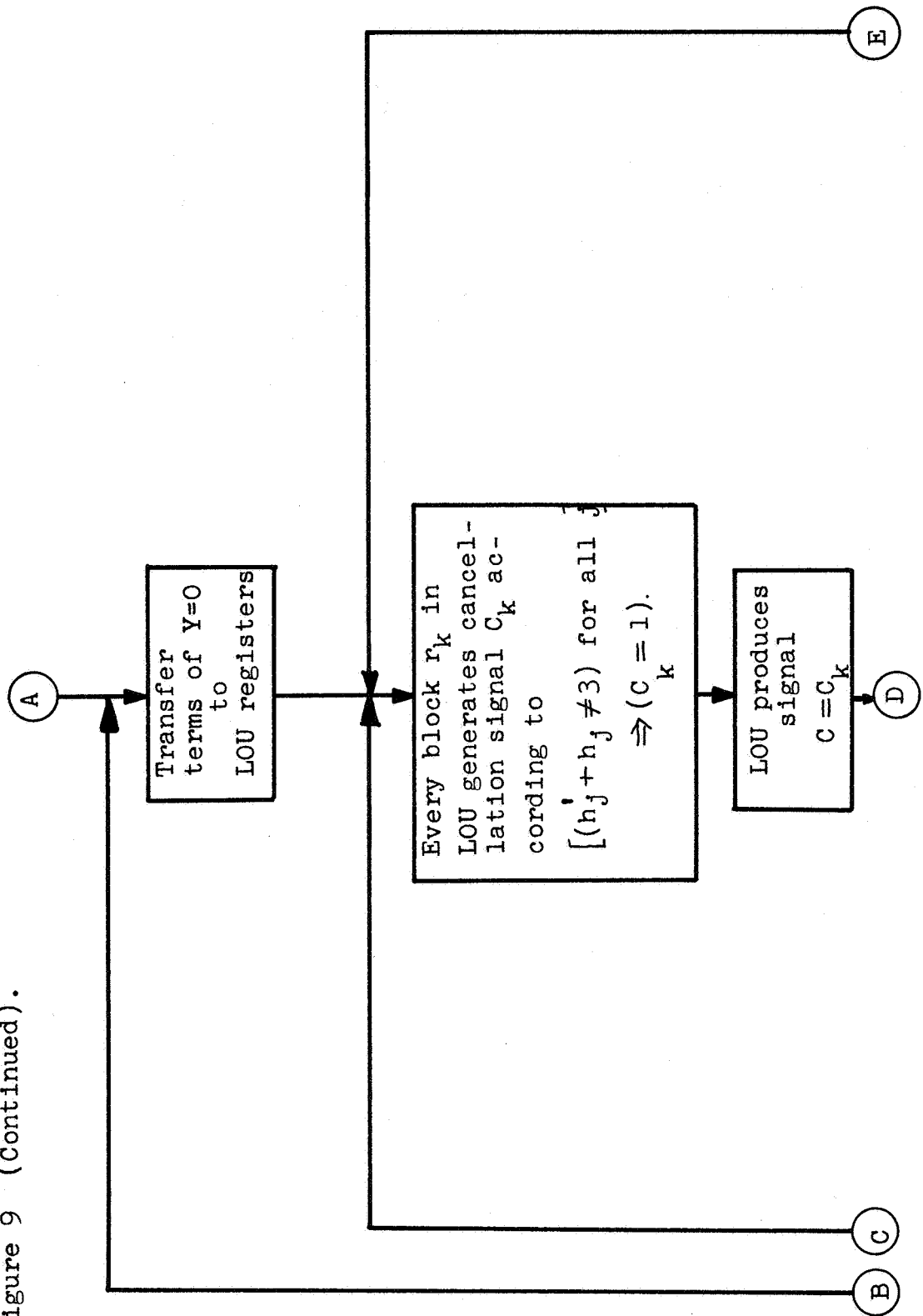


Figure 9 (Continued).



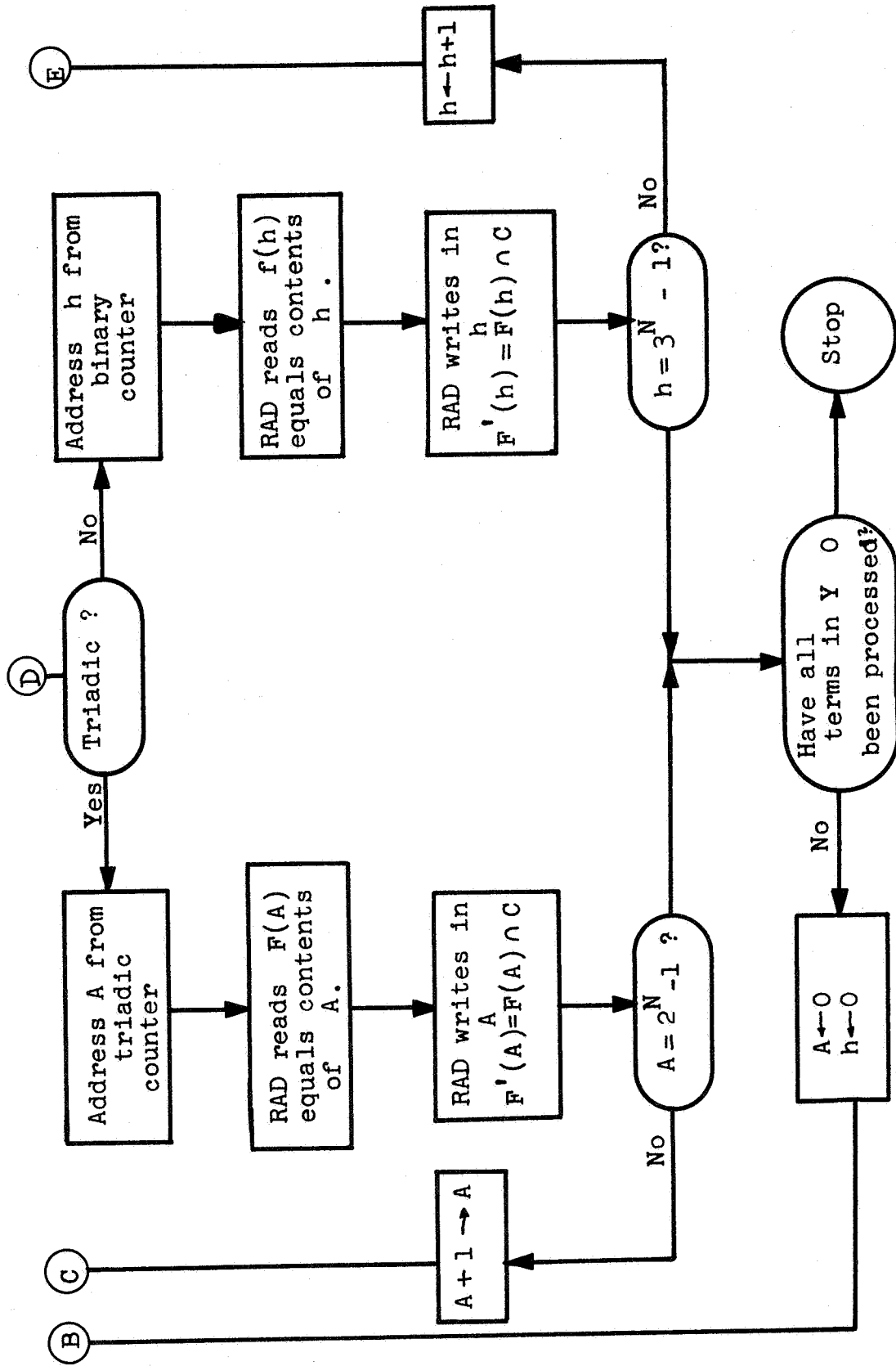


Figure 9. (Continued).

with zeros.

- 5) Triadic and Binary counters reset to zero
- 6) Indexing Block reset to zero.

3.3.2 MODE II: Non-prime-implicant determination

In this mode of operation, the Boolean Analyzer eliminates in the circulating memory (which is assumed to contain the implicants of a given Boolean function as the result of Mode I) those implicants which are not prime-implicants. The cancellation procedure is a direct implementation of Theorem 3 (Section 3.1) reformulated in the following condition

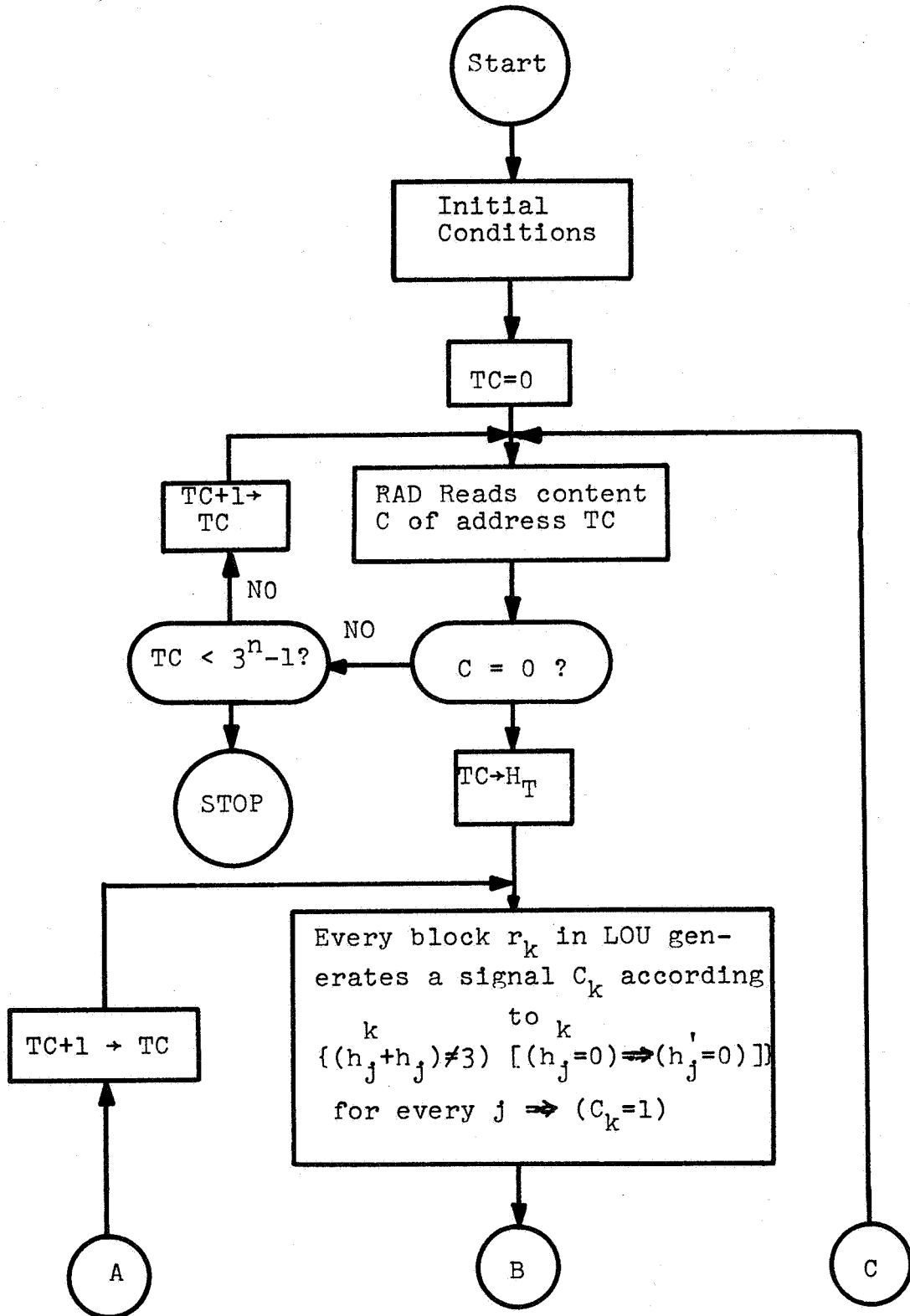
$$\underbrace{\{(h_j^k + h'_j \neq 3) \wedge [(h_j^k = 0) \Rightarrow (h'_j = 0)]\}}_{\text{for every } j} \Rightarrow (C_k = 1) \quad (26)$$

The procedure corresponding to MODE II is described in the flowchart of Figure 10. The following initial conditions are required:

- 1) Circulating Memory (used in Mode A) contains results of Mode I.
- 2) LOU is in Mode B
- 3) Snake Unit in Mode B
- 4) Indexes set to zero

In Table IV MODES I and II are summarized.

Figure 10. Flowchart corresponding to MODE II.



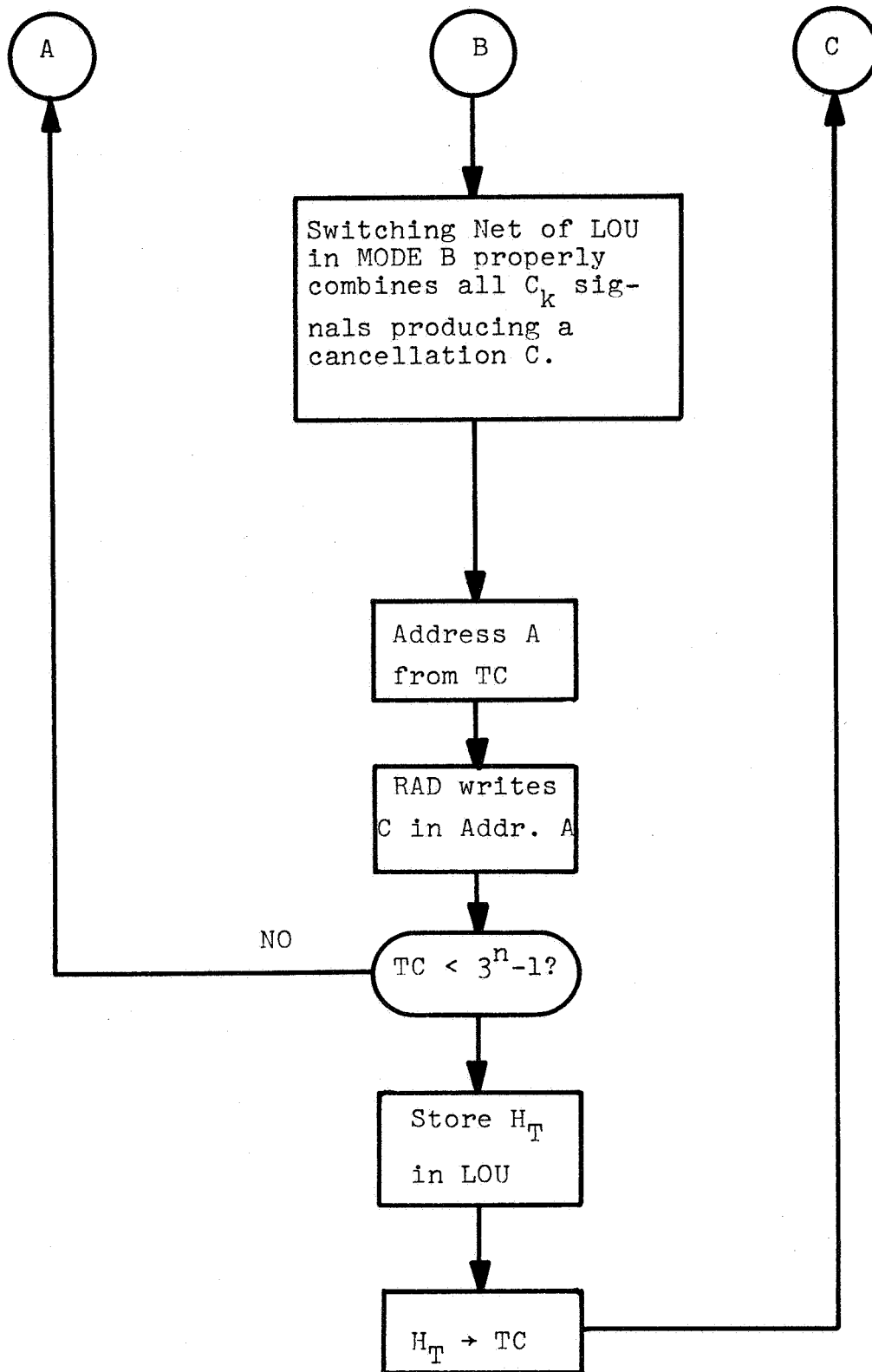


Figure 10(Continued). Flowchart corresponding to MODE II

TABLE IV
 MODES OF OPERATION OF THE BA

B. A. Mode	B. A. SUBSYSTEM				Mode Control
	Circulating Memory	Snake Unit	L O U Special Registers	Combinational Net	
MODE I	Mode A	Mode a	Triadic	Mode A	Triadic
	Mode B	Mode a	Binary	Mode A	Binary
MODE II	Mode A	Mode b	Triadic	Mode B	Triadic

CHAPTER IV
APPLICATIONS OF THE BOOLEAN ANALYZER SYSTEM

In Section 3.3 we presented the two fundamental modes of operation of the Boolean Analyzer Unit. In this Chapter the Boolean Analyzer System (composed by SIGMA 7 and Boolean Analyzer Unit) will be used to solve some problems in Boolean Algebra.

4.1 Solution of Systems of Boolean Equations

Svoboda's algorithm for solving Boolean Equations (Section 1.1.3) can be implemented using the Boolean Analyzer system. The following algorithmic steps are needed:

- Step #1. Transfer of terms in $Y = 0$ to the Logical Operation Unit.
- Step #2. Discriminant determination.
- Step #3. Computation of truth table of every existing solution.

Step #1 is executed by proper indexing of the flip-flops in LOU and by means of a WRITE DIRECT instruction. (For details, see reference 21).

Step #2 is computed using Boolean Analyzer in MODE I (binary mode).

Step #3 is executed by means of an adequate software program stored in SIGMA 7. This program called SOLDET decomposes the discriminant and gives the truth

tables of the set of functions corresponding to every solution. In Appendix I a flowchart and listing of this program is given.

In Figure 11 the algorithmic steps 1, 2, 3 are summarized.

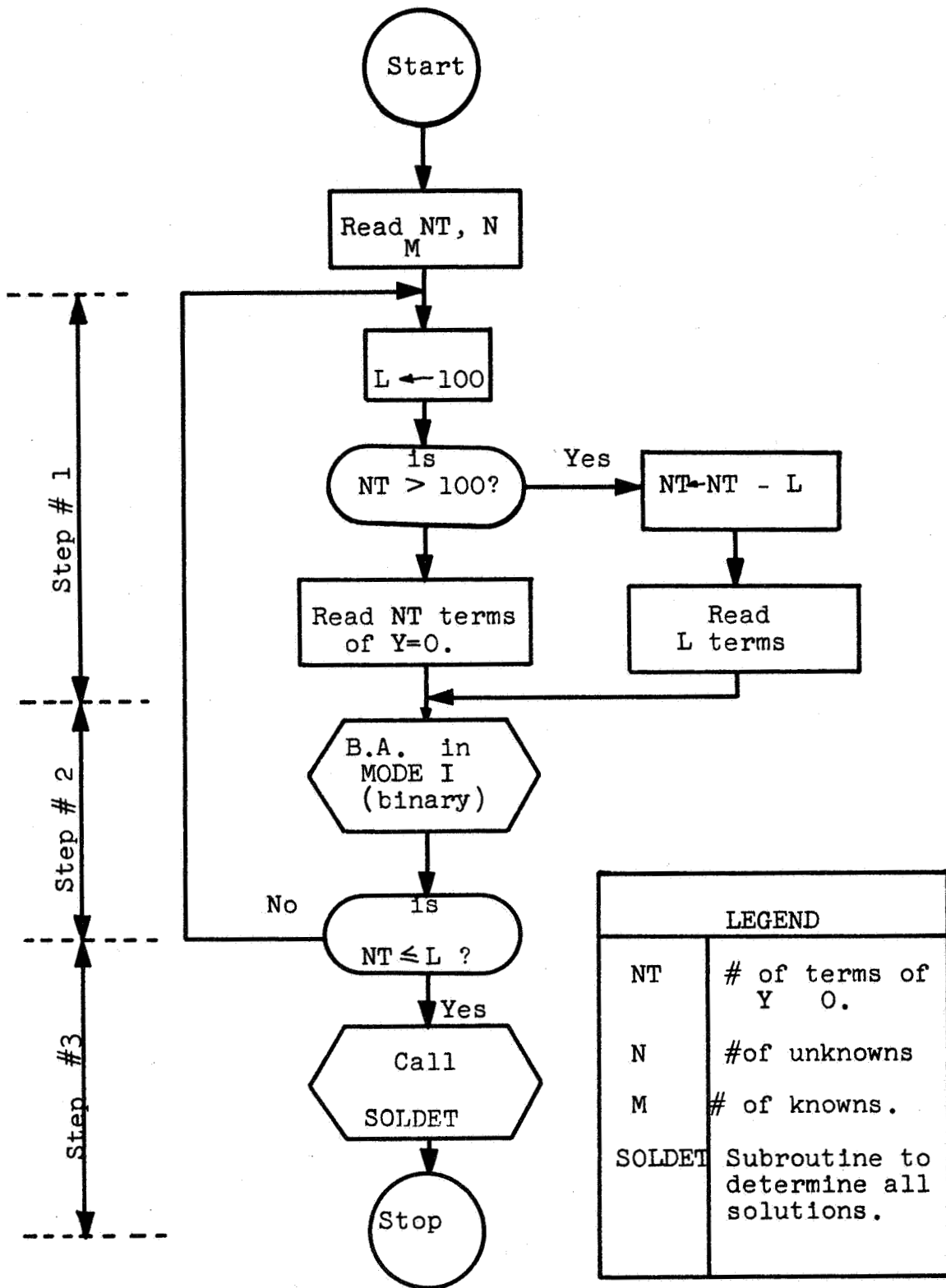


Figure 11. Flowchart for System of Boolean Equations.

4.2 Irredundant coverings of a Boolean Function

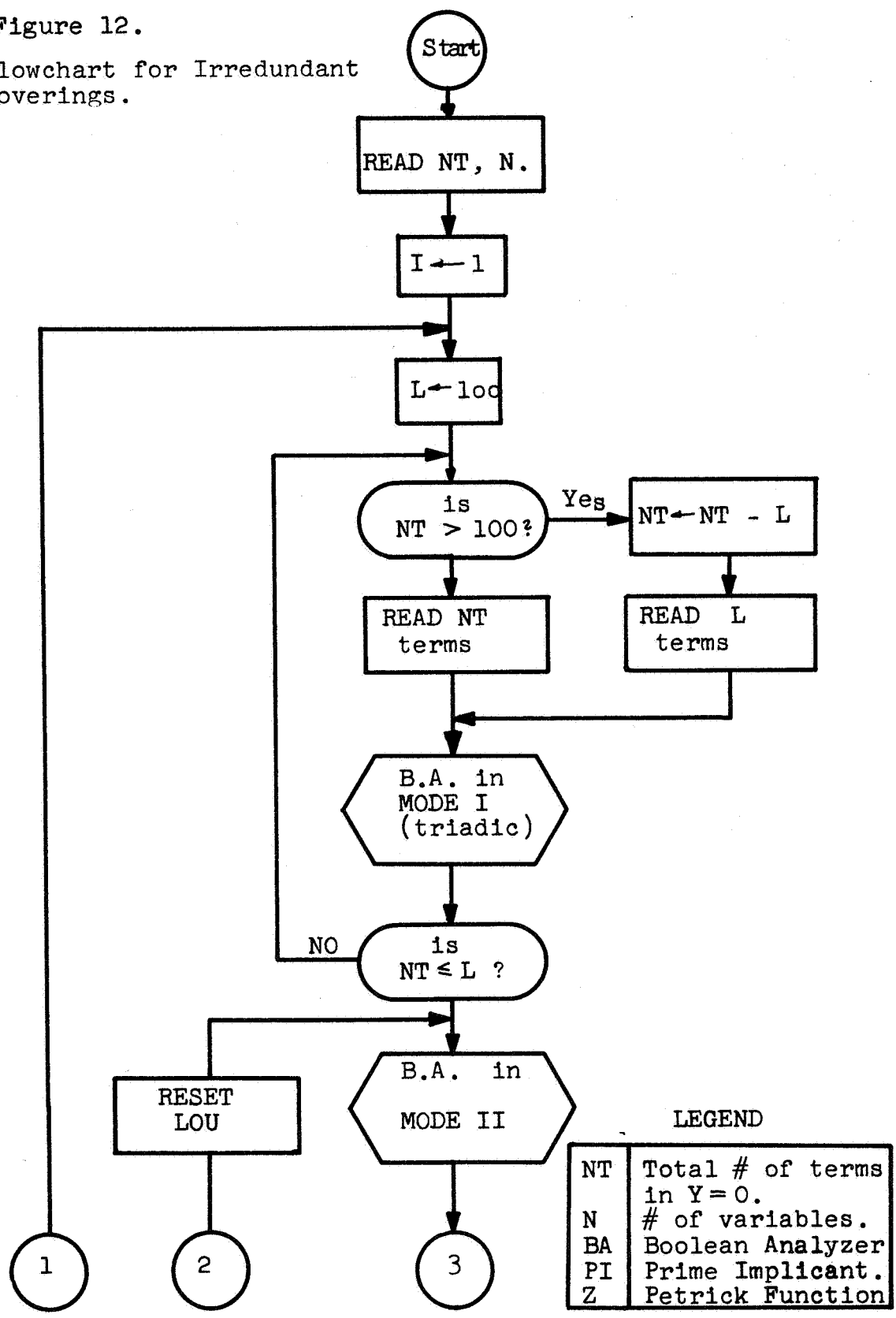
The problem of finding all irredundant coverings of a Boolean function y is a classical one and therefore it will not be presented here in detail. However, it is the purpose of this section to show how the Boolean Analyzer system may solve this problem.

The algorithmic steps are the following:

- Step #1 Transfer of terms of $Y = \bar{y}$ into the Logical Operation Unit of Boolean Analyzer.
- Step #2 Determination of all implicant terms of y . This step is executed using MODE I in triadic connexion.
- Step #3 Determination of all prime-implicant terms of y , by using MODE II.
- Step #4 The covering problem is formulated building a special function z , called Petrick function²⁹.
- Step #5 Steps 1, 2, 3 are applied to function $Z = \bar{z}$.
- Step #6 The prime-implicants of function z are printed, each one of them is an irredundant covering of the given function f .

The above algorithm is summarized in Figure 12.

Figure 12.
Flowchart for Irredundant Coverings.



LEGEND

NT	Total # of terms in $Y=0$.
N	# of variables.
BA	Boolean Analyzer
PI	Prime Implicant.
Z	Petrick Function

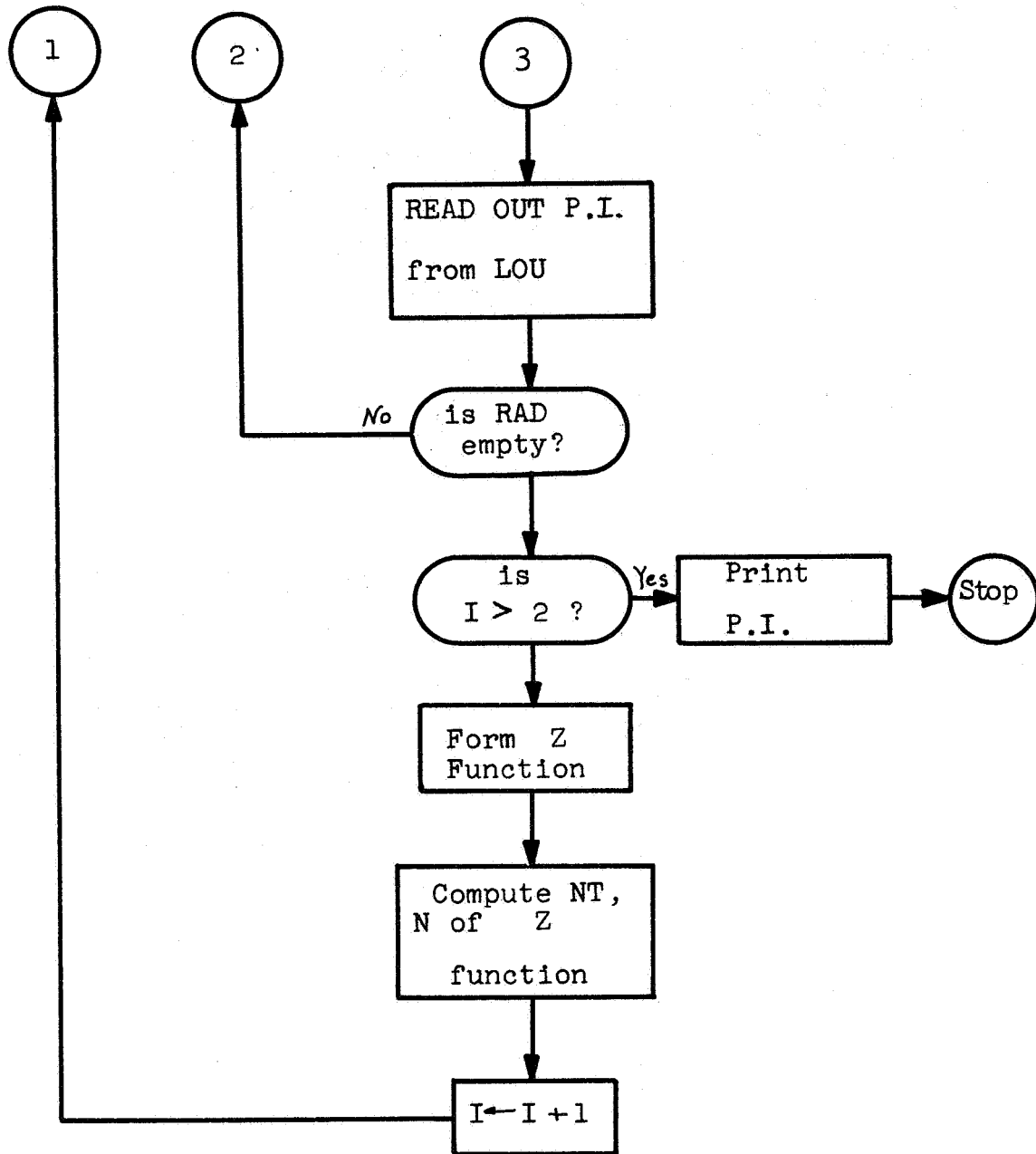


Figure 12. (Continued).

4.3 Synthesis of TANT Networks

4.3.1. Introduction

The synthesis of Three-level AND-NOT combinational networks with True inputs (TANT networks) has been studied by Marley and Ogden²⁴, McCluskey²⁵, Hellerman²⁶, Marley and Earle²⁷, and recently by Gimpel²⁸. This last author presents the first systematic synthesis and simplification method of TANT circuits. This method is very similar to the Quine-McCluskey minimization algorithm for two-level AND-OR logic.

Gimpel's method starts by generating a general form for the implicants of the given Boolean function f , called permissible-implicants of f , and in such a way that it is simple to derive the circuit configuration from this general form. From the set of permissible implicants, the set of prime-permissible implicants (which we shall call here generalized prime implicants) is determined and from this set those which realize the given function with a minimum number of NAND decision elements are selected.

In this section we propose another method of TANT synthesis based on the two modes of operation of Boolean Analyzer (Section 3.3).

The proposed method consists of calculating the ordinary prime implicants of the given function f and, from this set, to calculate all generalized prime implicants of f by solving a system of Boolean equations. The selection

of those generalized prime implicants which form the minimal TANT network is done by constructing a special Petrick function²⁹ Z and determining its prime implicants. The proposed method thus uses the two modes of operation of the Boolean Analyzer: the MODE II for the determination of the generalized prime implicants and MODE I in triadic mode for the determination of all minimal TANT networks.

4.3.2 DEFINITIONS AND THEOREMS

Consider the TANT circuit of Figure 13. It is easily verified that this circuit implements the function

$$f = x_2x_0 + x_0(\overline{x_1x_0}) + x_1\overline{x_2}(\overline{x_0x_1}) \quad (27)$$

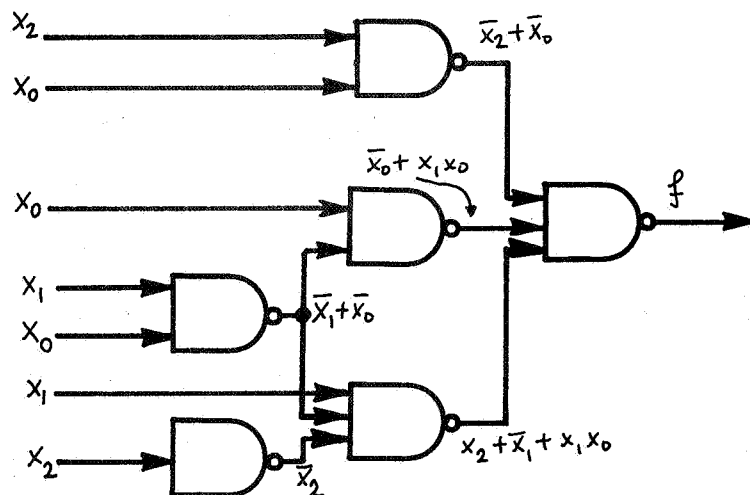


Figure 13
Example of TANT Circuit

This function is written in AND-OR form as follows:

$$f = x_2x_0 + x_0\bar{x}_1 + x_1\bar{x}_2\bar{x}_0$$

Form (27) of f presents a characteristic structure of sum of permissible terms (P-terms) whose general form may be defined in the following way:

Definition 1

Every time P , implicant of a Boolean function f of n variables and expressed in TANT for (27), may be written as product of two factors H and T

$$P = H.T \tag{28}$$

where $H = \ddot{x}_{n-1} \dots \ddot{x}_1 \dots \ddot{x}_0 \ddot{x}_1$ where \ddot{x}_i takes the values 1 or x_i exclusively ($i=0, \dots, n-1$)

$T = T_{m-1} T_{m-2} \dots T_1 T_0$ where $T_j = \dot{x}_{n-1} + \dots + \dot{x}_1 + \dots + \dot{x}_0$; \dot{x}_i takes the values 0 or \bar{x}_i exclusively. ($j = 0, \dots, m-1$), ($i=0, \dots, n-1$), $m \leq n$.

Example 1:

The term $P_1 = x_0(\overline{x_0x_1})$ has $H = x_0$, $T = T_1 = (\overline{x_0x_1}) = (\bar{x}_1 + \bar{x}_0)$.

The term $P_2 = x_1\bar{x}_2(\overline{x_0x_1})$ has $H = x_1$, and $T = T_2.T_1 = \bar{x}_2(\bar{x}_0 + \bar{x}_1)$.

Definition 2

A term P of the form (28) is said to be a

generalized prime implicant of f (GP-term) if an ordinary prime implicant of f (in the sense of the AND-OR logic) or a sum of any number of them is obtained when a P-term (28) is expanded using the rules of logical sum, product and complementation. The ordinary prime implicants of f will be denoted by OP.

Example 2:

The function f given in Figure 13 has the following ordinary prime implicants:

$$x_0\bar{x}_1, x_0x_2, \bar{x}_0x_1\bar{x}_2.$$

The terms P_1 and P_2 of Example 1 corresponding to the TANT form (27) of f are generalized prime implicants. Effectively, according to Definition 2,

$$P_1 = GP_1 = x_0(\overline{x_1x_0}) = x_0\bar{x}_1 + x_0\bar{x}_0 = x_0\bar{x}_1$$

$$P_2 = GP_2 = x_1\bar{x}_2(\overline{x_0x_1}) = x_1\bar{x}_2\bar{x}_1 + x_2\bar{x}_0\bar{x}_1 = x_2\bar{x}_0\bar{x}_1.$$

Definition 3: (Criterion of Minimal TANT network)

We shall say that a TANT network produces a certain given Boolean function f in a minimal form if no other TANT circuit exists which has less number of NAND decision elements.

Theorem 1:

According to the previous definition of minimal TANT network, every P-term (28) of a minimal TANT form of f is a generalized prime implicant.

Proof: Suppose that the minimal TANT expression of f is

$$P_1 + P_2 + \dots + P_1 + \dots + P_r$$

where each term P_i is of the form (28). We shall prove that P_i is formed by logical addition of any number of ordinary prime implicants OP of f . To prove this statement it is sufficient to prove that there is no minterm m_i of f such that

$$P_i = \sum_j OP_j + m_i \text{ with } m_i \notin \sum_j OP_j.$$

Effectively, according to Definition 1 the general form of P_i is $H_1 \cdot (T_1 T_2 \dots T_k)$; on the other hand, since OP_j is an ordinary prime implicant of f ,

$$\sum_j OP_j = H_1 \bar{x}_1 \dots \bar{x}_1 \dots \bar{x}_m, \quad x_i \notin H_1 \text{ for} \\ i = 1, \dots, m \quad m \leq n.$$

But $P_i = \sum_j OP_j + m_i$, therefore, the minterm m_i should have the same factor H_1 of uncomplemented variables that the terms OP_j and P_i . But m_i is a minterm and thus contains all complemented variables which are not contained in H_1 . These variables must be contained also in $\sum_j OP_j$ because, if not contained, or m_i is not a minterm or its H-factor of uncomplemented variables does not coincide with H_1 , implying that P_i is not of the form (28). Therefore, every term of a minimal TANT form is a generalized prime implicant.

The question is now how to determine the set of generalized prime implicants and, from this set, to select those that produce the function f according to Definition 3.

To form the set of generalized prime implicants we start from the set of ordinary prime implicants of f , as proven by the following theorem.

Theorem 2

Let $\{OP\}$ be the set of ordinary prime implicants of f and let $\{GP\}$ be the set of generalized prime implicants, then

$$(1) \quad \{OP\} \subset \{GP\}$$

$$(2) \quad GP_j = \sum_i OP_i \text{ for } OP_i \in \{OP\} \quad ; \quad GP_j \in \{GP\}.$$

The proof of these two propositions follows immediately from the definition of ordinary prime implicant of f and from Definition 2.

Based on theorem 2, it is clear that the set of ordinary prime implicants of f augmented with those P-terms, of the form (28), obtained by logical addition of any number of prime implicants OP_i contained in $\{OP\}$ constitutes the set of P-terms from which it is possible to generate all possible generalized prime implicants GP_j . This set of P-terms thus formed is called the BASE of the given function f . It will be denoted by B .

Note that once the set of ordinary prime implicants is obtained, only those with common H-factor will produce a new P-term of the base by logical addition. This is due to the special form of the P-terms (28).

Example 3: Consider the function of Figure 14

	$x_1 x_0$	00	01	10	11
x_2	0		●	●	
	1		●		●

Figure 14

TANT Synthesis Example

The set of ordinary prime implicants is

$$\{OP\} = \{x_0 \bar{x}_1, x_0 x_2, \bar{x}_0 x_1 \bar{x}_2\}.$$

Since there are no terms $OP_j \subset \{OP\}$ with common H-factor, the Base B of the function coincides with the set $\{OP\}$.

Thus

$$B = \{x_0 \bar{x}_1, x_0 x_2, \bar{x}_0 x_1 \bar{x}_2\}.$$

Example 4: Consider the function whose corresponding

Marquand map is

	$x_1 x_0$	00	01	10	11
x_2	0	●		●	●
	1			●	

The set $\{OP\}$ is $\{x_1\bar{x}_2, x_1\bar{x}_0, \bar{x}_0\bar{x}_2\}$.

In this case there are two terms OP_j with common H-factor: $x_1\bar{x}_2$ and $x_1\bar{x}_0$. These two terms are combined by logical addition producing the term $x_1(\overline{x_2x_0})$. The BASE is thus

$$B = \{x_1\bar{x}_2, x_1x_0, \bar{x}_0\bar{x}_2, x_1(\overline{x_2x_0})\}.$$

4.3.3 CALCULATION OF THE GENERALIZED PRIME IMPLICANTS

The generalized prime implicants of a given Boolean function f are constructed inserting in all possible ways the variables contained in the H-factor of each term of the base in its corresponding T-factors. The GP terms thus obtained which are identically equal to zero are disregarded.

Example 5: Consider the base $B = \{x_0\bar{x}_1, \bar{x}_0x_1\bar{x}_2, x_0x_2\}$.

From the term $x_0\bar{x}_1$ we obtain the term $GP_1 = x_0(\overline{x_0x_1})$.

From the term x_0x_2 no acceptable term is generated because those obtained: $x_0x_2(\bar{x}_0)$, $x_0x_2(\bar{x}_2)$, $x_0x_2(\overline{x_0x_2})$ are identically equal to zero. From term $x_1\bar{x}_0\bar{x}_2$ we obtain

$$GP_2 = x_1(\overline{x_1x_0})\bar{x}_2$$

$$GP_3 = x_1\bar{x}_0(\overline{x_2x_1})$$

$$GP_4 = x_1(\overline{x_1x_0})(\overline{x_2x_1})$$

The generalized prime implicants are:

$$GP_1 = x_0(\overline{x_1 x_0})$$

$$GP_2 = x_1(\overline{x_1 x_0})\bar{x}_2$$

$$GP_3 = x_1(\overline{x_2 x_1})\bar{x}_0$$

$$GP_4 = x_1(\overline{x_2 x_1})(\overline{x_0 x_1})$$

plus the P-terms of the base

$$GP_5 = x_0\bar{x}_1$$

$$GP_6 = \bar{x}_0 x_1 \bar{x}_2$$

$$GP_7 = x_0 x_2$$

This method of calculating the generalized prime implicants is preferred when the synthesis is done manually. However, the automatization by standard computer programming of this method is laborious because it is required to handle variable-length words and many shift operations to insert the variables of the H-factor in the T-factor. Also, the identification of H- and T-factors belonging to the same P-term is not easy to implement. For these reasons, and due to the fact that the Boolean Analyzer is capable of solving large systems of Boolean equations, we studied the possibility of transforming the problem of generalized prime implicants generation to the solution of a system of Boolean equations.

Before describing the set up of this system of equations we need the following theorem.

Theorem 3

If a T-factor of a P-term belonging to the base B of a given Boolean function f of n variables has $m(m \leq n)$ complemented variables, the GP-terms obtained from this P-term has m distinct T_j factors ($j = 1, \dots, m$).

Proof: Let the term $OP_1 = x_0 \dots x_1 \cdot \bar{x}_j \dots \bar{x}_k$, where $(r+s)$

$$\leq n, \overbrace{x_0, \dots, x_1}^s \neq \overbrace{x_j, \dots, x_k}^r, \text{ and let}$$

$$P = OP_j, P \in B \text{ and } OP_j \in \{OP\}.$$

Suppose that it is possible to generate from this P-term a generalized prime implicant of the form

$$GP_1 = x_0 \dots x_1 \cdot \bar{x}_j \dots \bar{x}_k (T_q) \quad (29)$$

According to Theorem 2, the factor T_q has to have the form

$$T_q = (a\bar{x}_0 + b\bar{x}_1 + \dots + h\bar{x}_1 + A\bar{x}_j + \dots + K\bar{x}_k)$$

(where $a, b, \dots, h, A, \dots, K$ are logical variables), because, if it will not have this form, or P does not belong to the base B, or GP_1 (29) is not a P-term of the form (28). But, suppose that the term GP_1 (29) is accepted as a generalized prime implicant of the minimal TANT expression, then it is evident that the circuit thus obtained will use an additional gate to synthesize T_q as compared with the

circuit obtained by accepting the term P instead of GP_1 . In the case that the factor T_q is available already in the synthesized circuit because it may belong to a necessary GP_j - term ($j \neq i$), it is easily seen that it is also preferred the term P instead of GP_1 because by using P an input at the decision element that realizes the term GP_1 is saved. Therefore, the term GP_1 (29) with $m + 1$ factors T does not introduce any additional information in the synthesis process and thus is not considered as an effective generalized prime implicant of f.

Theorem 3 gives the general form of the equation to be solved to obtain the acceptable generalized prime implicants. Effectively, given a term P_i of the base,

$$P_i = x_0 \overset{s}{\dots} x_i \cdot \bar{x}_j \overset{r}{\dots} \bar{x}_k, \quad (r + s) \leq n$$

$$x_0, \dots, x_i \neq \bar{x}_j, \dots, \bar{x}_k$$

every GP_j - term derived from P_i , $P_i \in B$, is a solution of the equation

$$x_0 \overset{s}{\dots} x_i \cdot \bar{x}_j \overset{r}{\dots} \bar{x}_k = x_0 \overset{s}{\dots} x_i \cdot (a_1 \bar{x}_0 + b_1 \bar{x}_1 + \dots + h_1 \bar{x}_{n-1}) \dots (a_r x_0 + b_r x_1 + \dots + h_r x_{n-1}) \quad (30)$$

where a_q, b_q, \dots, h_q for $q = 1, 2, \dots, r$ are logical variables.

The equation (30) may be considered as a system of Boolean equations with the unknowns a_q, b_q, \dots, h_q ($q=1, \dots, r$). This equation may be written in the form $Y = 0$ as required

for the Boolean Analyzer. The discriminant is obtained using Mode II of the Boolean Analyzer which is used to obtain all possible solutions of (30). The values of the unknowns substituted in the right hand side of (30) give the terms P which form the set of generalized prime implicants originated by the term P of the base.

Example 6: Consider the term $P_1 = x_1 \bar{x}_0 \bar{x}_2$. The set of generalized prime implicants derived from this term was already obtained in Example 5. The corresponding equation (30) for P_1 is:

$$x_1 \bar{x}_0 \bar{x}_2 = x_1 (a_1 \bar{x}_2 + b_1 \bar{x}_1 + c_1 \bar{x}_0) (a_2 \bar{x}_2 + b_2 \bar{x}_1 + c_2 \bar{x}_0) \quad (31)$$

Let us form the discriminant of this equation; i.e., in the space of $2^6 \cdot 2^3 = 2^9$ (6 unknowns and 3 constants) let us determine the minterms for which equation (31) is satisfied. This discriminant is shown in Figure 15.

The interesting solutions to our problem are those which are invariant with respect to the values of the constants x_0, x_1, x_2 ; i.e., those which satisfy (31) no matter the values of the constants x_0, x_1, x_2 . If the discriminant is organized as we have done in Figure 15: rows representing the space of unknowns and columns the space of constants, the desired solutions may be read directly from the discriminant, because each solution

x_2 0 0 0 0 1 1 1 1
 x_1 0 0 1 1 0 0 1 1
 x_0 0 1 0 1 0 1 0 1

	c_2	b_2	a_2	c_1	b_1	a_1
0	0	0	0	0	0	0
1	0	0	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	0	0	1	0	0
5	0	0	0	1	0	1
6	0	0	0	1	1	0
7	0	0	0	1	1	1
8	0	0	1	0	0	0
9	0	0	1	0	0	1
10	0	0	1	0	1	0
11	0	0	1	0	1	1
12	0	0	1	1	0	0
13	0	0	1	1	0	1
14	0	0	1	1	1	0
15	0	0	1	1	1	1
16	0	1	0	0	0	0
17	0	1	0	0	0	1
18	0	1	0	0	1	0
19	0	1	0	0	1	1
20	0	1	0	1	0	0
21	0	1	0	1	0	1
22	0	1	0	1	1	0
23	0	1	0	1	1	1
24	0	1	1	0	0	0
25	0	1	1	0	0	1
26	0	1	1	0	1	0
27	0	1	1	0	1	1
28	0	1	1	1	0	0
29	0	1	1	1	0	1
30	0	1	1	1	1	0
31	0	1	1	1	1	1
32	1	0	0	0	0	0
33	1	0	0	0	0	1
34	1	0	0	0	1	0
35	1	0	0	0	1	1
36	1	0	0	1	0	0
37	1	0	0	1	0	1
38	1	0	0	1	1	0
39	1	0	0	1	1	1
40	1	0	1	0	0	0
41	1	0	1	0	0	1
42	1	0	1	0	1	0
43	1	0	1	0	1	1
44	1	0	1	1	0	0
45	1	0	1	1	0	1
46	1	0	1	1	1	0
47	1	0	1	1	1	1
48	1	1	0	0	0	0
49	1	1	0	0	0	1
50	1	1	0	0	1	0
51	1	1	0	0	1	1
52	1	1	0	1	0	0
53	1	1	0	1	0	1
54	1	1	0	1	1	0
55	1	1	0	1	1	1
56	1	1	1	0	0	0
57	1	1	1	0	0	1
58	1	1	1	0	1	0
59	1	1	1	0	1	1
60	1	1	1	1	0	0
61	1	1	1	1	0	1
62	1	1	1	1	1	0
63	1	1	1	1	1	1

Figure 15.
Discriminant of
Equation (31).

corresponds to every row containing only dots (a dot means that for this minterm equation (31) will hold). The solutions of (31) have been marked with an arrow in Figure 15 for easy identification, and are listed below:

identifier	Space of unknowns	terms obtained by substitution in (5)
	$c_2 b_2 a_2 c_1 b_1 a_1$	
12	0 0 1 1 0 0	$x_1(\bar{x}_0) (\bar{x}_2)$
14	0 0 1 1 1 0	$x_1(\overline{x_0 x_1}) (\bar{x}_2)$
28	0 1 1 1 0 0	$x_1(\bar{x}_0) (\overline{x_2 x_1})$
30	0 1 1 1 1 0	$x_1(\overline{x_0 x_1}) (\overline{x_2 x_1})$
33	1 0 0 0 0 1	$x_1(\bar{x}_2) (\bar{x}_0)$
35	1 0 0 0 1 1	$x_1(\overline{x_1 x_2}) (\bar{x}_0)$
49	1 1 0 0 0 1	$x_1(\bar{x}_2) (\overline{x_1 x_0})$
51	1 1 0 0 1 1	$x_1(\overline{x_0 x_1}) (\overline{x_0 x_2})$

From this set of solutions we eliminate those that are repeated because the logical multiplication is commutative. Thus, solution #12 is identical to solution #33, #14 to #49, #28 to #35 and #30 to #51. Finally we obtain the following generalized prime implicants

$$GP_2 = x_1(\overline{x_1 x_0}) \bar{x}_2$$

$$GP_3 = x_1 \bar{x}_0 (\overline{x_2 x_1})$$

$$GP_4 = x_1(\overline{x_0 x_1}) (\overline{x_2 x_1})$$

$$GP_6 = x_1 \bar{x}_0 \bar{x}_2$$

which coincide with those previously obtained. (We have kept here the same subindices than on Example 5).

The method of generating the generalized prime implicants through the solution of systems of Boolean equations is very long if done manually. Note, for example, that for functions with only 3 variables a logical space of 9 variables is required. The method, however, is completely general and is specially adapted to the Boolean Analyzer. The general purpose computer (in our case the Sigma 7 computer) will generate the terms of the base B, and, for each term of this base, it will generate an equation like (30) but in the form $Y = 0$. The terms of Y will be inserted in the logical operation unit of the Boolean Analyzer and the discriminant will be obtained in the circulating memory (in our case a RAD Storage System). Sigma 7 will read out this discriminant selecting those identifiers from the space of unknowns which verify (30). The solutions thus obtained are stored in the memory of Sigma 7 and the system proceeds with the next term of the base. At the end of this process, all generalized prime implicants of f will be stored in Sigma 7.

In order to show the automatization possibility of the above described method, a program was derived for

Sigma 7, called SBEV3 (Systems of Boolean Equations Version 3) which simulates the Boolean Analyzer in this particular application. The program SBEV3 was written in FORTRAN IV language and for functions f of three variables only*. In Appendix II a list of the instructions of this program and the output data obtained for the following function f of three variables is given:**

	$x_1 x_0$	00	01	10	11
x_2	0			•	•
	1		•	•	

** This function is taken from an example presented by Gimpel in Reference 28.

* A more general program is presented in Chapter 5.

Generalized Prime Implicants		
Terms of the Base		Calculated GP Terms
Number	Expression	
		$x_2 x_0 (\overline{x_1 x_0})$
		$x_2 x_0 (\overline{x_1 x_2})$
1	$x_2 x_0 \overline{x_1} \dots \dots \dots$	$x_2 x_0 (\overline{x_0 x_1 x_2})$
2	$x_1 \overline{x_0} \dots \dots \dots$	$x_1 (\overline{x_0 x_1})$
3	$x_1 (\overline{x_0 x_2}) \dots \dots \dots$	$x_1 (\overline{x_0 x_1 x_2})$
4	$x_1 \overline{x_2} \dots \dots \dots$	$x_1 (\overline{x_1 x_2})$

Table V
GENERALIZED PRIME IMPLICANTS

To interpret properly the output data given by the program SBEV3, the following terminology is used:

BASE TERM NUMBER = : refers to the ordering number of the base.

LIST OF TERMS OF THE FUNCTION $Y = 0$: refers to the terms of Equation (30) written in form $Y = 0$.

IDENTIFIER OF THE SOLUTION: refers to the identifier of the logical space

of unknowns for which a solution exists.

In our case, since there are 3 unknowns, this logical space corresponds to the unknowns a,b,c, exclusively and in correspondence with the constants x_2, x_1, x_0 respectively.

Thus, for the first term of the base (see Table V) : $x_0 x_2 \bar{x}_1$, the program SBEV3 gives the following solutions:

identifiers	a,b,c	T-factor	GP-term
2	0 1 0	x_1	$x_2 x_0 \bar{x}_1$
3	0 1 1	$(\overline{x_1 x_0})$	$x_2 x_0 (\overline{x_1 x_0})$
6	1 1 0	$(\overline{x_2 x_1})$	$x_2 x_0 (\overline{x_2 x_1})$
7	1 1 1	$(\overline{x_2 x_1 x_0})$	$x_2 x_0 (\overline{x_2 x_1 x_0})$

For the second term of the base: $x_1 \bar{x}_0$ the solutions are

1	0 0 1	\bar{x}_0	$x_1 \bar{x}_0$
3	0 1 1	$(\overline{x_1 x_0})$	$x_1 (\overline{x_1 x_0})$

For the third term of the base: $x_1 (\overline{x_0 x_2})$ the solutions are

identifier	a,b,c	T-factor	GP-term
5	1 0 1	$(\overline{x_2 x_0})$	$x_1(\overline{x_2 x_0})$
7	1 1 1	$(\overline{x_2 x_1 x_0})$	$x_1(\overline{x_2 x_1 x_0})$

For the fourth term of the base: $x_1 \overline{x_2}$ the solutions are:

4	1 0 0	$\overline{x_2}$	$x_1 \overline{x_2}$
6	1 1 0	$(\overline{x_2 x_1})$	$x_1(\overline{x_2 x_1})$

The prime implicants obtained with the program SBEV3 coincide with those of Table V.

4.3.4 THE SELECTION OF GENERALIZED PRIME IMPLICANTS

In order to obtain a minimal TANT synthesis of a Boolean function f not all generalized prime implicants are required. This is also the case of the AND-OR logic of two levels. The problem is now to select those generalized prime implicants which produce the function f according to the criterion of minimality given in Definition 3.

In a TANT expression, and due to the fact that there exists a third logical level, it is possible to use the same elements of the third logical level in several

inputs of elements of the other levels, and therefore the selection of GP-terms consists of two problems:

- (1) The selection of the minimum number of GP-terms which cover the function.
- (2) From all TANT expressions obtained select those which use the least number of NAND elements, i.e. obtain maximum "sharing" of T-factors.

The first problem is a classical one and has been solved by McCluskey³⁰ using tables or algebraically by Petrick²⁹ by means of a logical function Z which expresses the condition of covering of the function f .

The second problem has been solved by Luccio and Grasselli³¹ using a special table, called CC-table (Cover and Closure Table) but in connection with the problem of simplification of the number of internal states of a sequential circuit. In this section we propose a similar method as the one using CC-tables but we differ in the fact that our table is used to generate two special Petrick functions that in combination solve the covering problem.

In the following lines we describe systematically our investigations in solving this covering problem.

We apply first the Petrick function to the example

on Table V. For this purpose a table is constructed whose columns represent the minterms of the function f and the rows the generalized prime implicants. We enter a dot in those places of the table where the generalized prime implicant of a row covers a minterm of a column. This table is the following:

	$x_2x_1x_0$	$x_2x_1\bar{x}_0$	$x_2\bar{x}_1x_0$	$x_2x_1\bar{x}_0$
$GP_1 = x_2x_0\bar{x}_1$			•	
$GP_2 = x_2x_0(\overline{x_1x_0})$			•	
$GP_3 = x_2x_0(\overline{x_1x_2})$			•	
$GP_4 = x_2x_0(\overline{x_2x_1x_0})$			•	
$GP_5 = x_1\bar{x}_0$	•			•
$GP_6 = x_1(\overline{x_1x_0})$	•			•
$GP_7 = x_1(\overline{x_2x_0})$	•	•		•
$GP_8 = x_1(\overline{x_2x_1x_0})$	•	•		•
$GP_9 = x_1\bar{x}_2$	•	•		
$GP_{10} = x_1(\overline{x_2x_1})$	•	•		

Table VI
COVERING OF MINTERMS

If the proposition "All minterms of the function are covered" is identified with the Boolean variable Z, this variable may be expressed as a function of the GP_i ($i = 1, \dots, 10$) terms as follows:

$$Z = (GP_5 + GP_6 + GP_7 + GP_8 + GP_9 + GP_{10}) \cdot (GP_7 + GP_8 + GP_9 + GP_{10}).$$

$$(GP_1 + GP_2 + GP_3 + GP_4) \cdot (GP_5 + GP_6 + GP_7 + GP_8) \quad (32)$$

Each term gives us a covering of f. Evidently those terms of Z with the least number of GP-terms will give the most economical implementation. In other words, the ordinary prime implicants of Z are the irredundant coverings of f. The expansion of (32) gives

$$Z = GP_1 \cdot GP_7 + GP_2 \cdot GP_7 + GP_3 \cdot GP_7 + GP_4 \cdot GP_7 + GP_1 \cdot GP_8 + GP_2 \cdot GP_8 + GP_3 \cdot GP_8 + GP_4 \cdot GP_8 \quad (33)$$

where each term is already a prime implicant of Z.

Any term of (33) that we may choose for the TANT circuit has the same number of GP-terms. This selection, however, does not suffice to obtain a minimum number of gates implementation because it is possible to use an

output of the third logical level (T-factor) as inputs to several gates of the second level. The inspection of TABLE VI shows immediately that GP_4 and GP_8 have the same T-factor: $(\overline{x_0 x_1 x_2})$ and since $GP_4 \cdot GP_8$ is a term of (33) these two GP-terms give a minimal TANT implementation of F. The corresponding circuit is given in Figure 16.

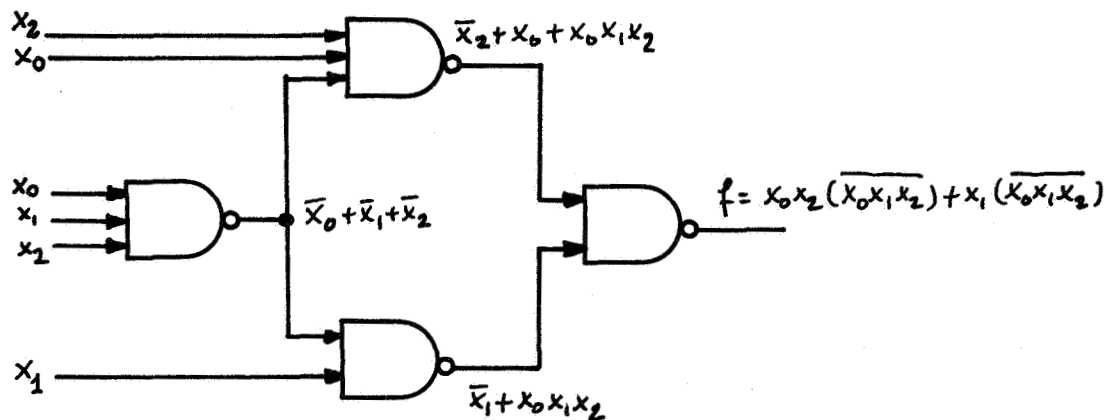


Figure 16
TANT Circuit Corresponding to Table V

The visual inspection of TABLE VI is not in general that easy for problems of large numbers of variables, and since we are interested in automatic methods of selection we investigate the following two alternatives:

- (1) To construct another table based on a function like (33)
- (2) To add logical conditions to the Petrick function to include the second selection process.

Let us consider both alternatives separately.

The First alternative consists of constructing another table (TABLE VII) in which each row corresponds to each term of the Petrick function previously obtained (33) and with each column corresponding to every T_j factor of the GP-terms contained in (33). If a dot is entered in those places of the tables where a T_j -factor is present in a GP-term, the answer to our selection problem is the row(s) with the least number of dots. If several rows exist with the least number of dots, all give a minimal solution according to Definition 3. However, if the criterion of minimum total number of inputs to the gates of the circuit is considered, we will choose from the table those T_j factors with the least number of literals.

To illustrate this process consider the previous example and let us build a table (TABLE VII) in the way indicated with the terms of (33).

	\bar{x}_1	$(\overline{x_1 x_0})$	$(\overline{x_1 x_2})$	$(\overline{x_2 x_1 x_0})$	$(\overline{x_2 x_0})$
GP ₁ . GP ₇	●				●
GP ₂ . GP ₇		●			●
GP ₃ . GP ₇			●		●
GP ₄ . GP ₇				●	●
GP ₁ . GP ₈	●			●	
GP ₂ . GP ₈		●		●	
GP ₃ . GP ₈			●	●	
GP ₄ . GP ₈				●	

Table VII
T-FACTORS SELECTION

The row with the least number of points is the last one which corresponds to the term GP₄ . GP₈ as expected.

Following, we present another example of synthesis of a function of 4 variables.

Example 7: Let f be taken as given by the map of Figure 17.

		$x_1 x_0$			
		00	01	10	11
$x_3 x_2$	00	●	●		●
	01				
	10	●	●		
	11		●		●

The terms of the base

are:

$$B = \{x_0 \bar{x}_2 \bar{x}_3, \bar{x}_1 \bar{x}_2, x_0 x_3 \bar{x}_1, x_0 x_2 x_3\}$$

Figure 17

Truth Table of Example 7

Generalized prime
implicants

Minterms

		0	1	3	8	9	13	15
$GP_1 = x_0 \bar{x}_2 \bar{x}_3$			●	●				
$GP_2 = x_0 (\overline{x_0 x_2}) \bar{x}_3$			●	●				
$GP_3 = x_0 \bar{x}_2 (\overline{x_3 x_0})$			●	●				
$GP_4 = \bar{x}_1 \bar{x}_2$		●	●		●	●		
$GP_5 = x_0 x_3 \bar{x}_1$						●	●	
$GP_6 = x_0 x_3 (\overline{x_1 x_3})$						●	●	
$GP_7 = x_0 x_3 (\overline{x_1 x_0})$						●	●	
$GP_8 = x_0 x_3 (\overline{x_1 x_0 x_3})$						●	●	
$GP_9 = x_0 x_2 x_3$							●	●

The corresponding Petrick function is:

$$Z = GP_4(GP_1 + GP_2 + GP_3 + GP_4) \cdot (GP_1 + GP_2 + GP_3) \cdot GP_4 \cdot (GP_4 + GP_5 + GP_6 + GP_7 + GP_8) \cdot (GP_5 + GP_6 + GP_7 + GP_8 + GP_9) \cdot GP_9$$

which is reduced to:

$$Z = GP_4 \cdot GP_9(GP_1 + GP_2 + GP_3) = GP_4 GP_9 GP_1 + GP_4 GP_9 GP_2 + GP_4 GP_9 GP_3$$

The associated table to this function is:

	\bar{x}_1	\bar{x}_2	\bar{x}_3	$(\overline{x_0 x_2})$	$(\overline{x_3 x_0})$	
$GP_4 \cdot GP_9 \cdot GP_1$	●	●	●			minimal solution
$GP_4 \cdot GP_9 \cdot GP_2$	●	●	●	●		
$GP_4 \cdot GP_9 \cdot GP_3$	●	●			●	minimal solution

There exist two solutions with the same number of NAND elements:

$$f = GP_4 + GP_9 + GP_1 = \bar{x}_1 \bar{x}_2 + x_0 x_2 x_3 + x_0 \bar{x}_2 \bar{x}_3 \quad (34)$$

$$f = GP_4 + GP_9 + GP_3 = \bar{x}_1 \bar{x}_2 + x_0 x_2 x_3 + x_0 \bar{x}_2 (\overline{x_3 x_0})$$

In Figures 18 and 19 we show the TANT circuits realizing forms (34) and (35).

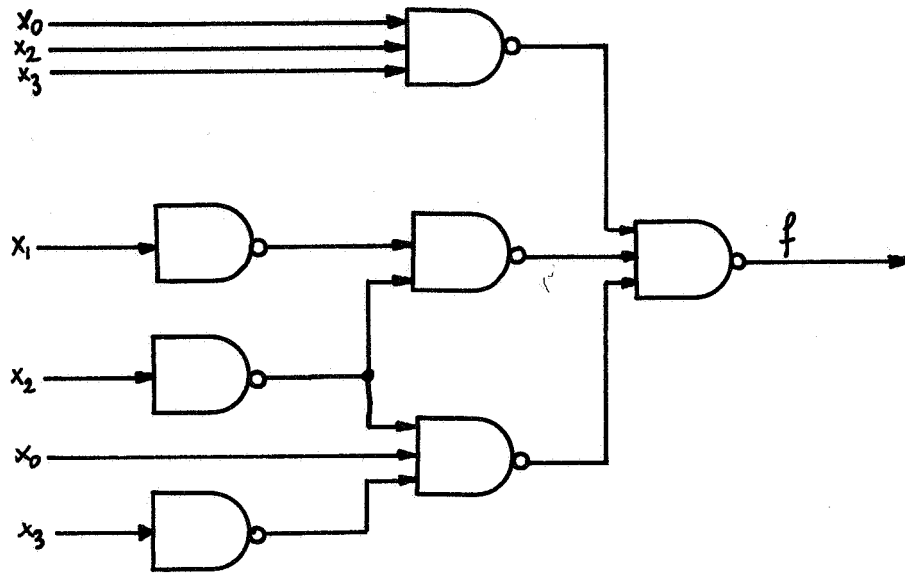


Figure 18
TANT Circuit Corresponding to Example 7,
First Solution.

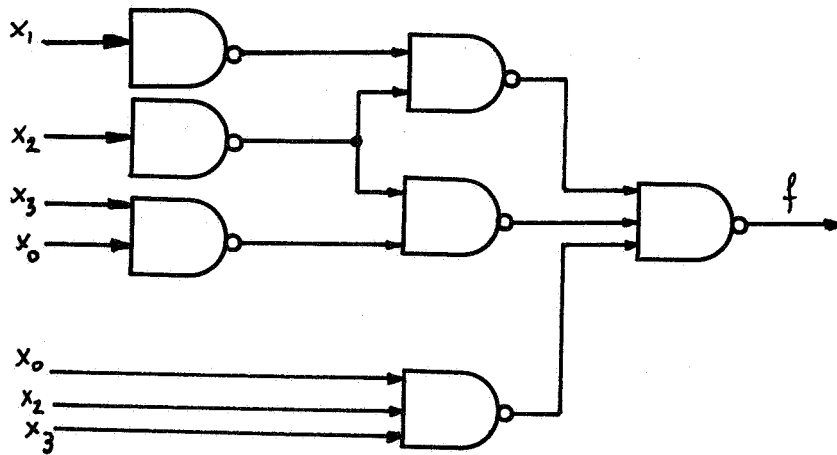


Figure 19
TANT Circuit Corresponding to Example 7,
Second Solution.

The Second alternative in the selection of GP-terms is to add to the Petrick function Z (19) a set of logical conditions (equations) that together with Z give the solution of the problem.

If we expand the original Table V from which Z (33) was derived by adding new entries each one corresponding to a T_j -factor belonging to every GP-term and if it is marked with a dot those places of the expanded-side when the T_j -factor belongs to a certain GP-term (as was done in TABLE VII), new equations may be generated that together with Z express the logical conditions for minimal covering.

This process was illustrated in the previous example. The extended table of GP-terms with the T_j -factors is given in TABLE VIII.

Generalized Prime Implicants	minterms															\bar{x}_1	\bar{x}_2	\bar{x}_3	(\bar{x}_2x_0)	(\bar{x}_3x_0)	(\bar{x}_3x_1)	(\bar{x}_1x_0)	$(\bar{x}_1x_0x_3)$
	0	1	3	8	9	13	15																
$GP_1 = x_0\bar{x}_2\bar{x}_3$		•	•																				
$GP_2 = x_0(\bar{x}_0x_2)\bar{x}_3$		•	•																				
$GP_3 = x_0\bar{x}_2(\bar{x}_3x_0)$		•	•											•									
$GP_4 = \bar{x}_1\bar{x}_2$	•	•		•																			
$GP_5 = x_0x_3\bar{x}_1$					•																		
$GP_6 = x_0x_3(\bar{x}_1x_3)$					•										•								
$GP_7 = x_0x_3(\bar{x}_1x_0)$					•												•						
$GP_8 = x_0x_3(\bar{x}_1x_0x_3)$					•														•				
$GP_9 = x_0x_2x_3$																				•			

TABLE VIII . SELECTION TABLE. Q_1 Q_2 Q_3 Q_4 Q_5 Q_6 Q_7 Q_8

From the columns of minterms one deduces the Petrick function Z which is the same as (33) and by using the columns of T_j -factors one obtains the logical equations which express logically that the indicated T_j -factors belong to the corresponding GP-terms. The T_j -factors are named here with the letter Q (see TABLE VIII). Thus the set of equations obtained is:

$$Z = GP_4 GP_9 GP_1 + GP_4 GP_9 GP_2 + GP_4 GP_9 GP_3 \quad (33)$$

$$GP_1 = Q_2 \cdot Q_3$$

$$GP_5 = Q_1$$

$$GP_2 = Q_3 \cdot Q_4$$

$$GP_6 = Q_6$$

$$GP_3 = Q_2 \cdot Q_5$$

$$GP_7 = Q_7$$

$$GP_4 = Q_1 \cdot Q_2$$

$$GP_8 = Q_8$$

If (36) is substituted in (33), the result obtained is:

$$Z = Q_1 Q_2 GP_9 Q_3 + Q_1 Q_2 Q_3 Q_4 GP_9 + Q_1 Q_2 Q_5 GP_9$$

The first and the last terms contain the same least number of variables and, therefore, are the solutions of the covering problem. Thus,

$$f = GP_4 + GP_9 + GP_1$$

$$f = GP_4 + GP_9 + GP_3$$

which coincide with (34) and (35) previously obtained.

From the point of view of automation, the second alternative seems to be more convenient because once the ordinary prime implicants of Z are obtained it suffices to do the above mentioned substitution (i.e. Equations (36) in (32)) and this process may be done easily in the general purpose computer (Sigma 7).

If Figure 20 we show a schematic flow chart of the automated proposed process of TANT synthesis as carried out by the system Sigma 7 - Boolean Analyzer.

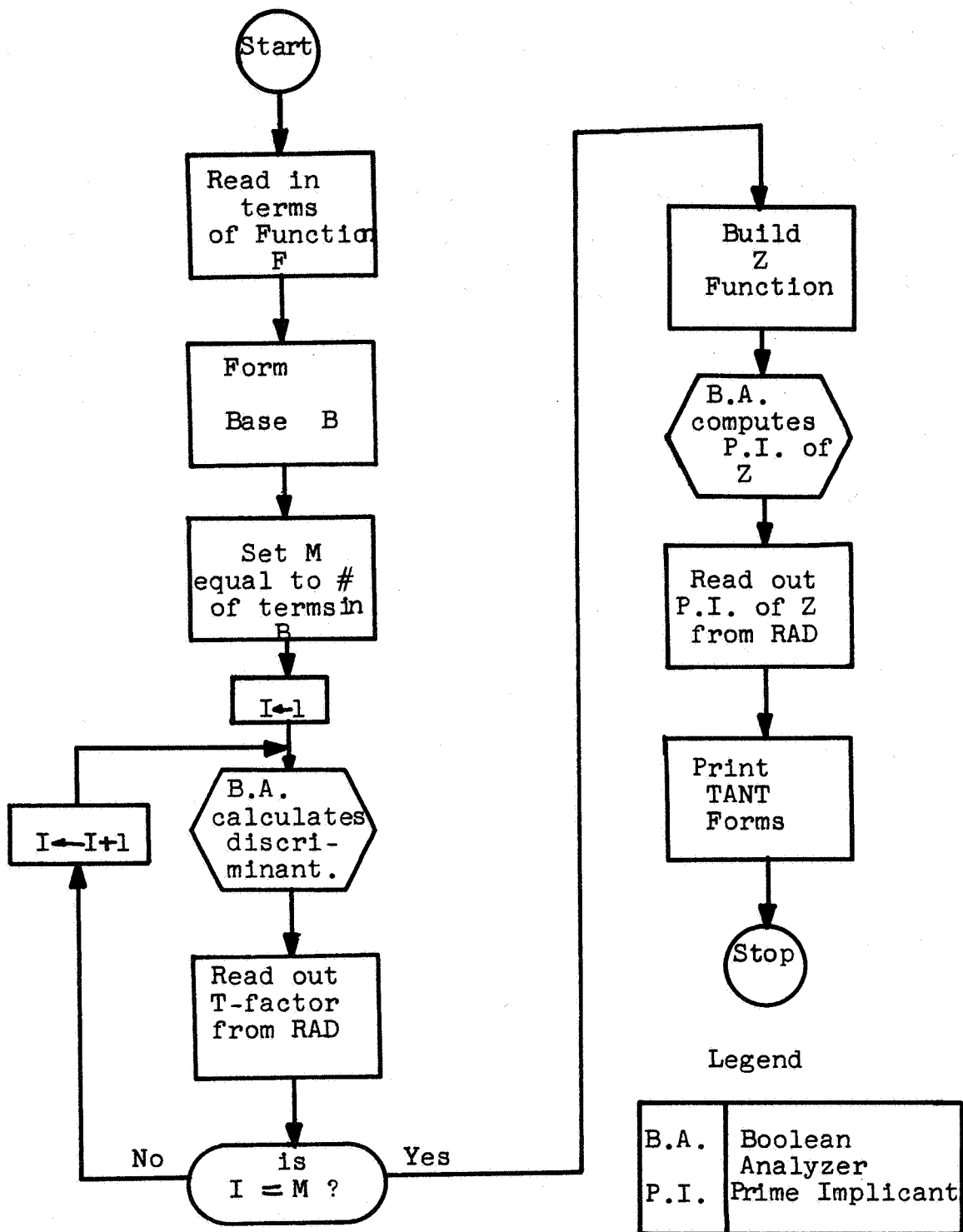


Figure 20.
Flowchart of the TANT Synthesis using the B.A.

4.4 A General Synthesis Method of Logical Nets using fixed inventory of integrated circuit modules

4.4.1 Synthesis of Single Output Logical Nets

4.4.1.1 Introduction

The synthesis of combinational or switching circuits is a major task in digital computer design. Somewhere along the design process a set of Boolean (logical) equations are stated by the designer which describe the properties of the digital system. The designer is then faced with the task of implementing this set of equations using, in general, a restricted inventory of integrated circuits which constitute the basis building blocks. The structure of this inventory imposes restrictions on the use of the commonly known AND-OR two-level logical circuit design techniques, such as Quine-McCluskey, etc., because if followed, the synthesis thus obtained is not necessarily the optimum with respect to the particular inventory. A question, therefore arises how to proceed or what general method should be followed to obtain a given Boolean Function by using any general building block and what are the conditions under which this task is possible with the given particular inventory?

First, the inventory should be adequate. This means that the set of available integrated circuits should be such that any given Boolean Function can be synthesized. It has been proven that inventories having only NAND gates³², or only NOR gates³², or as recently proven by Patt³³ invento-

ries having only a fixed macro module (for instance, one called WOS), are sufficient to synthesize any Boolean Function. In this section we shall only consider inventories of integrated circuits with only one type of logical elements.

Second, the synthesis algorithm to be followed should be independent of the complexity or simplicity of the used inventory. In other words, the synthesis method should be applicable with no modification to any given type of inventory. This requirement will give maximum generality to the synthesis algorithm.

Third, the number of steps of the algorithm should be finite and a certain criterion of "optimum" should be followed at the different steps to satisfy the specific designer's constraints. This criterion should not affect the synthesis algorithm but rather the existing function of the inventory. Here an interaction with the synthesis procedure by the designer is desirable.

In this section we introduce a general algorithm which satisfies the requirements mentioned above and produces an adequate implementation of a logical net provided the designer establishes an "adequate" set of design criteria. Thus we may say that an acceptable synthesis is obtained depending upon the design criteria imposed on the existing inventory functions and upon the completeness of this inventory.

The synthesis method used here is based on Svoboda's Algorithm for Solving Boolean Equations (Section 1.1.3).

Every level of the logical net is synthesized by solving a system of Boolean Equations and by choosing (according to the designer's criteria) the acceptable solutions. This method can be partly or totally mechanized on a digital computer. Its implementation using the Boolean Analyser is of particular interest as we shall see.

4.4.1.2 The Synthesis Algorithm.

Suppose that a certain Boolean function f has to be implemented with a certain building block or integrated circuit module which produces a logical function F .

The problem is to find the internal structure of the "black box" of Figure 21 which assume contains no memory devices.

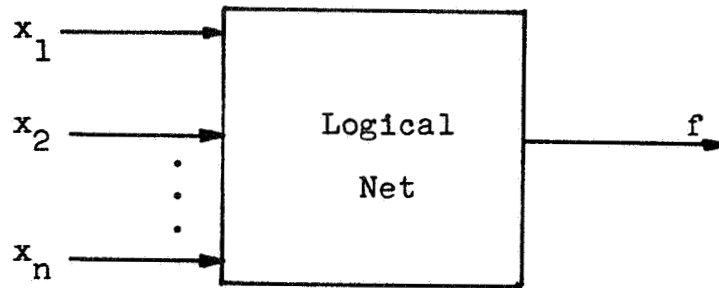


Figure 21
Combinational Logic Net.

Since the logical net (Figure 21) contains only one kind of modules, it is clear that the output f has to

be also the output of a module*.

Therefore the "black box" of Figure 21 must have the structure shown in Figure 22.

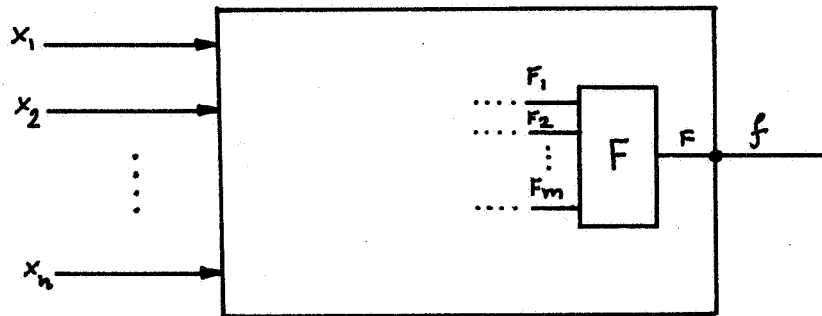


Figure 22.
Combinational Logic Net,
Showing Last Logic Level.

and thus the following Boolean relation exists :

$$f = F \quad (36)$$

We observe that F is a given function of the m finite number of inputs F_1, F_2, \dots, F_m of the given module and that f is a given function of the n input logical variables x_1, x_2, \dots, x_n . Thus Equation (36) has the form

$$f(x_1, x_2, \dots, x_n) = F(F_1, \dots, F_m) \quad (37)$$

The problem now consists of solving the Boolean equation (37) for F_1, F_2, \dots, F_m as functions of x_1, x_2, \dots, x_n to

* We do not consider here the case where f degenerates in one of the input variables. This is a trivial synthesis problem because no module is required.

determine the input functions to the last module of the logical net. These input functions F_1 are then considered as output functions of other modules F (Figure 22) whose inputs are determined by the same process. The synthesis procedure ends when the input functions to the modules at a particular stage become equal to any of the input logical variables x_1, x_2, \dots, x_n , its complements, if these are available, or fixed logical levels, "0", "1".

In general, the solution of equation (37) is not unique and, therefore, a criterium is required to reduce the total number of existing solutions to adequate ones.

The determination of this criterium is left to the experience and specific requirements of the designer. For example, suppose that the designer would like to have a logical net with minimum number of logical stages meaning optimum delay. In this case he may instruct the machine to select those solutions of (37) which are :

- (1) fixed logical levels : logical "1" or logical "0"
- (2) input variables
- (3) function directly produces by the module being used.

In the following applications we shall consider this criterium.

We should mention here that the chosen criterium of optimum does not affect the basic steps of the synthesis algorithm. This characteristic enables the easy experimentation with

different criteria of optimality, because these affects exclusively the existing functions of the inventory.

4.4.1.3 Synthesis of Logical Nets Using NAND Logic

To illustrate the synthesis algorithm described in the previous section, a particular application to NAND logical nets is presented here. We assume that both true and complement variables are available.

The inventory is composed, for example, of all functions produced by three-input-NAND-gates. If we use the criterium of minimum number of gates in the circuit, the acceptable solutions of equation (37) will be one of the following three categories :

- (1) functions implemented by a single 3-input-NAND-gate
- (2) input variables (complemented and uncomplemented)
- (3) fixed logical levels : logical "0" or logical "1" .

In order to conveniently evaluate every possible solution, we first catalog the inventory functions according to an identifier, and second we assign a weighting factor to each input-function F_1 according to its category.

The identifier of each function F_1 is obtained by computing the decimal equivalent of the binary string representing the truth values of the corresponding function.

For example, the identifier of function

$$f = \text{NAND}(x_1, x_2) = \bar{x}_1 + \bar{x}_2$$

is equal to 119. Effectively,

	128	64	32	16	8	4	2	1	
x_3 :	1	1	1	1	0	0	0	0	
x_2 :	1	1	0	0	1	1	0	0	
x_1 :	1	0	1	0	1	0	1	0	
Truth values f :	(0	1	1	1	0	1	1	1)	$_2 = (119)_{10}$.

Similarly, the identifier corresponding to $f = \text{NAND}(x_1, x_2, x_3)$ is 127.

The functions of the inventory cataloged according to its ID number are given in the following table :

TABLE IX NAND FUNCTIONS OF 3. INPUTS


<u>Identifier</u>	<u>function:</u> f	<u>inputs:</u> 
0	0	directly available
15	\bar{x}_3	directly available
51	\bar{x}_2	directly available
63	$\bar{x}_2 + \bar{x}_3$	$x_2 \ x_3 \ 1$
85	\bar{x}_1	directly available
95	$\bar{x}_1 + \bar{x}_3$	$x_1 \ x_3 \ 1$
119	$\bar{x}_1 + \bar{x}_2$	$x_1 \ x_2 \ 1$
127	$\bar{x}_1 + \bar{x}_2 + \bar{x}_3$	$x_1 \ x_2 \ x_3$
170	x_1	directly available
175	$x_1 + \bar{x}_3$	$\bar{x}_1 \ x_3 \ 1$
187	$x_1 + \bar{x}_2$	$\bar{x}_1 \ x_2 \ 1$
191	$x_1 + \bar{x}_2 + \bar{x}_3$	$\bar{x}_1 \ x_2 \ x_3$
204	x_2	directly available
207	$\bar{x}_3 + x_2$	$x_3 \ \bar{x}_2 \ 1$
221	$\bar{x}_1 + x_2$	$x_1 \ \bar{x}_2 \ 1$
223	$\bar{x}_1 + x_2 + \bar{x}_3$	$x_1 \ \bar{x}_2 \ x_3$
240	x_3	directly available
243	$\bar{x}_2 + x_3$	$x_2 \ \bar{x}_3 \ 1$
245	$\bar{x}_1 + x_3$	$x_1 \ \bar{x}_3 \ 1$
247	$\bar{x}_1 + \bar{x}_2 + x_3$	$x_1 \ x_2 \ \bar{x}_3$
248	$x_1 + x_2$	$\bar{x}_1 \ \bar{x}_2 \ 1$

TABLE IX CONTINUES

<u>Identifier</u>	<u>function:</u> f	<u>inputs:</u>		
		1	2	3
249	$x_1 + x_2 + \bar{x}_3$	\bar{x}_1	\bar{x}_2	x_3
250	$x_1 + x_3$	\bar{x}_1	\bar{x}_3	1
251	$x_1 + \bar{x}_2 + x_3$	\bar{x}_1	x_2	\bar{x}_3
252	$x_2 + x_3$	\bar{x}_2	\bar{x}_3	
253	$x_1 + x_2 + x_3$	x_1	\bar{x}_2	\bar{x}_3
254	$x_1 + x_2 + x_3$	\bar{x}_1	\bar{x}_2	\bar{x}_3
255	1	directly available		

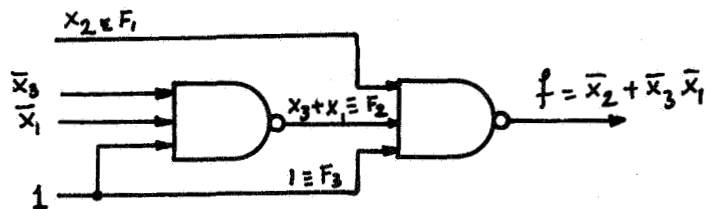
The assignement of weighting factors to each function F_1 is done using the following convention : Assume that the module being used has m inputs,

If function F_1 is a constant logical level, then weight-factor = 0

If function F_1 is an input variable in true (or complement form, when available), then weight-factor = 1

If function F_1 is a function directly obtainable with a single m -input-NAND-element, then weight factor = $(m + 1)$.

For example, the function $f = \bar{x}_2 + \bar{x}_3 \bar{x}_1$ may be implemented with the following circuit



where F_1, F_2, F_3 (input functions to the last logical level) have weight-factors 1, 4, 0 respectively.

Since each set of input functions F_1 is a possible solution of Equation (37), we may associate to each solution a weight factor which is the sum of the weights associate to its corresponding input function F_1 . Thus the weight associated to the solution : $F_1 = x_2, F_2 = x_3 + x_1, F_3 = 1$ in the previous example is $1 + 4 + 0 = 5$.

This solution-weight is useful to determine which of the possible solutions is more acceptable. Effectively, the so-

lution with lowest weight will be optimum according to the above mentioned criterium.

With the help of the identifiers of the functions contained in the inventory and the weight associated to each possible solution, the synthesis algorithm proper is easily executed in the following steps :

1st Step : Solve Eqn. (37) and compute the total number of existing solutions. Here the B.A. operating in MODE II may be used to compute the discriminant of Eqn. (37).

2nd Step : If the total number of existing solutions is not equal to zero, then compute the 1st solution and its identifier. If it is equal to zero, PRINT message ' no solution' and STOP.

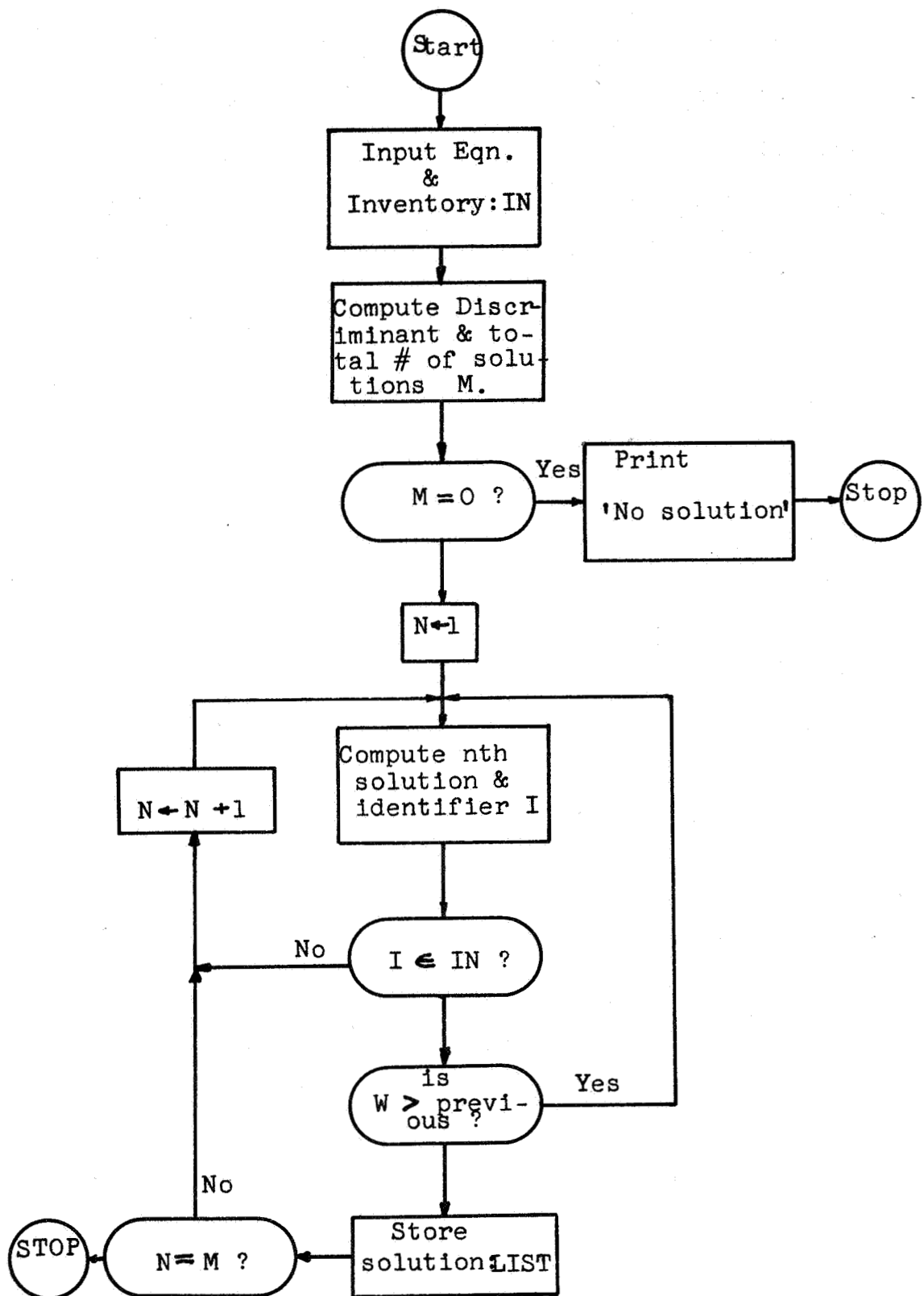
3rd Step : If the identifier computed in Step 2 is contained in the inventory, then compute solution-weight and print : Solution number, identifier of the solution, solution-weight and truth tables of corresponding input functions.

4th Step : Is the solution considered in Step 3 the last one? If yes, STOP. If no, consider next solution as first solution and repeat Steps 2 to 4 .

In the cases where only solutions with lowest weight are in-

teresting, the printing part of Step 3 is suppressed and substituted by a checking process which will store in an array named LIST the solutions with lowest weight. This option is specified by means of a data card to the computer program implementing the above algorithmic steps.

The following flowchart displays the steps of the synthesis algorithm.



A computer program, called SBEV4 (Appendix III) implementing the above flowchart has been written by generalization of the SBEV3 program for solving Boolean equations (Appendix II) and in such a way that when the inventory of functions is empty the program computes all existing solutions of the system of Boolean equations. However, when inventory functions are part of the input data (designer's constraints) then the program generates the acceptable solutions, and, if so specified, will print out only those solutions with lowest weight. To illustrate the synthesis algorithm, consider the following example.

Example : Synthesize with 3-input-NAND-gates the function

$$f = x_1 x_2 + \bar{x}_3 .$$

The general form of the output function of a 3-input-NAND-gate is

$$F = \bar{F}_1 + \bar{F}_2 + \bar{F}_3$$

therefore we solve the Boolean equation

$$x_1 x_2 + \bar{x}_3 = \bar{F}_1 + \bar{F}_2 + \bar{F}_3 \quad (38)$$

for F_1, F_2, F_3 as functions of x_1, x_2, x_3 .

Following Svoboda's algorithm (Section 1.1.3) we obtain the discriminant :

y \ x	0	1	2	3	4	5	6	7
0	1	1	1	1	0	0	0	1
1	1	1	1	1	0	0	0	1
2	1	1	1	1	0	0	0	1
3	1	1	1	1	0	0	0	1
4	1	1	1	1	0	0	0	1
5	1	1	1	1	0	0	0	1
6	1	1	1	1	0	0	0	1
7	0	0	0	0	1	1	1	0

$$x = 2^0 x_1 + 2^1 x_2 + 2^2 x_3$$

$$y = 2^0 F_1 + 2^1 F_2 + 2^2 F_3$$

$$7 \times 7 \times 7 \times 7 = 1 \times 1 \times 1 \times 7 = 16807 \text{ solutions.}$$

The total number of possible solutions is 16807. The acceptable ones are those with functions F_1, F_2, F_3 contained in Table IX (functions of the inventory).

In Appendix III we show the output given by the SBEV4 computer program corresponding to solutions with the lowest weight found equal to 5.

To verify the completeness of the synthesis method, let us show that the following solution (Fig. 23) which has been derived manually using standard techniques³⁸ is included in the set of solutions obtained by the SBEV4 program.

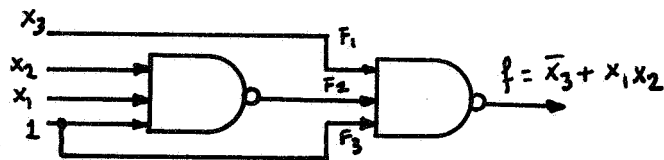


Figure 23. NAND Synthesis Example.

Effectively, the set of input functions to the last logical level (Fig.23) is :

$$\begin{aligned} F_1 &= x_3 \\ F_2 &= \bar{x}_1 + \bar{x}_2 \\ F_3 &= 1 \end{aligned} \quad (39)$$

The corresponding truth tables of these functions are

	x_1	0	1
$x_3 x_2$	0	0	0
	0	1	0
	1	0	1
	1	1	1

F_1

	0	1
	1	1
	1	0
	1	1
	1	0

F_2

	0	1
	1	1
	1	1
	1	1
	1	1

F_3

which are also obtained by the selection of the encircled non-zero elements (Fig.24) in the discriminant of Eqn. (38).

y	x	0	1	2	3	4	5	6	7
0		1	1	1	1	0	0	0	1
1		1	1	1	1	0	0	0	1
2		1	1	1	1	0	0	0	1
3		1	1	1	1	0	0	0	1
4		1	1	1	1	0	0	0	1
5		1	1	1	1	0	0	0	①
6		①	①	①	1	0	0	0	1
7		0	0	0	0	①	①	①	0

$$\begin{aligned} x &= 2^0 x_1 + 2^1 x_2 + 2^2 x_3 \\ y &= 2^0 F_1 + 2^1 F_2 + 2^2 F_3 \end{aligned}$$

Figure 24
Discriminant of Eqn. (38)

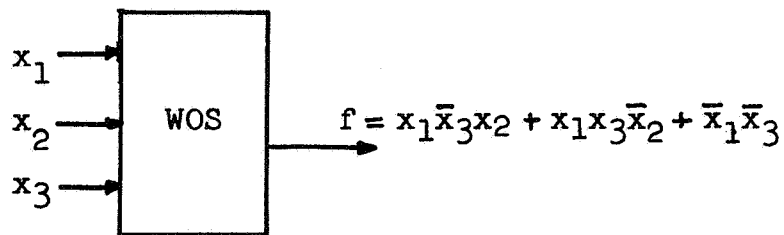
Looking at TABLE IX we find that functions F_1 , F_2 , F_3 given in (39) have identifiers 240, 119 and 255 respectively. This solution corresponds to solution number 7 given by SBEV⁴ (Appendix III).

4.4.1.4 Synthesis of Logical Nets Using WOS-Modules

In a recent paper, Patt³³ proposes a new building block for the implementation of logical nets. This module is called WOS-module (well-organized sequence-module) and is proven to be universal, i.e., any Boolean function may be generated by a circuit containing only WOS-modules. Patt also develops a synthesis method for this particular logic and shows that more economical circuits are obtained than with NAND logic elements.

In this section we shall illustrate our synthesis method as it applies to this particular case. The first step is to build the proper inventory functions. We shall assume that only true variables are available and that circuits with minimum number of modules are desired. This constraint will define the functions of the inventory.

A 3-input WOS-module is a logical circuit which performs the following Boolean function f

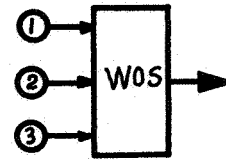


The module is asymmetric meaning that a permutation of the input variables causes a different output function.

The inventory of available functions is composed of: (1) functions generated by a single WOS-module, (2) only true input variables, (3) fixed logical levels.

The identifiers corresponding to these functions are 0, 3, 5, 10, 12, 15, 17, 34, 45, 48, 51, 57, 60, 63, 68, 75, 80, 85, 89, 90, 95, 99, 101, 102, 119, 153, 165, 170, 175, 187, 195, 204, 207, 221, 240, 243, 245, 255, (TABLE X). Note that there are 38 available functions in this inventory instead of 28 (TABLE IX) which was the case using NAND logic (Section 4.4.1.3). This means that we have enlarged by 10 the number of acceptable functions and thus we may expect differences in the circuits obtained with 3-input WOS-modules when compared with those obtained using 3-input NAND elements. This difference has already been shown by Patt in reference 33.

TABLE X
WOS FUNCTIONS OF 3-INPUTS



Identifier	Function	Input configuration			Algebraic equation
		1	2	3	
0	empty				0
3	0,1	x_3	x_2	x_2	$\bar{x}_3 \bar{x}_2$
		x_2	x_3	x_3	
5	0,2	x_3	x_1	x_1	$\bar{x}_3 \bar{x}_1$
		x_1	x_3	x_3	
10	1,3	x_1	x_3	1	$x_1 \bar{x}_3$
12	2,3	x_2	x_3	1	$x_2 \bar{x}_3$
15	0,1,2,3	1	1	x_3	\bar{x}_3
		1	x_3	1	
		x_3	1	1	
		x_3	1	x_3	
		0	x_3	x_3	
		x_3	x_3	x_3	
		x_3	x_3	x_3	
17	0,4	x_2	x_1	x_1	$\bar{x}_2 \bar{x}_1$
		x_1	x_2	x_2	
34	1,5	x_1	x_2	1	$x_1 \bar{x}_2$
45	1,3,4,6	x_1	x_2	x_3	$x_1 \bar{x}_3 x_2 + x_1 \cdot$ $x_3 \bar{x}_2 + \bar{x}_1 \bar{x}_3$
48	4,5	x_3	x_2	1	$x_3 \bar{x}_2$
51	0,1,4,5	x_2	1	x_2	\bar{x}_2

TABLE X CONTINUES

identifrier	Function	Input configuration			Algebraic equation
		1	2	3	
57	0,3,4,5	0	x_2	x_2	$x_1\bar{x}_2x_3+x_1x_2$ $\bar{x}_3+\bar{x}_1\bar{x}_2$
		x_2	x_2	x_2	
		x_1	x_3	x_2	
60	2,3,4,5	1	x_2	x_3	$\bar{x}_3x_2+x_3x_2$
63	0,1,2,3, 4,5	x_2	x_3	x_2	$\bar{x}_3+\bar{x}_2$
		x_3	x_2	x_3	
68	2,6	x_2	x_1	1	$x_2\bar{x}_1$
75	0,1,3,6	x_2	x_1	x_3	$x_2\bar{x}_3x_1+$ $x_2x_3\bar{x}_1+\bar{x}_2\bar{x}_3$
80	4,6	x_3	x_1	1	$x_3\bar{x}_1$
85	0,2,4,6	0	0	x_1	\bar{x}_1
		x_1	0	0	
		0	1	x_1	
				
89	0,3,4,6	x_1	x_1	x_1	$\bar{x}_2\bar{x}_1+x_2\bar{x}_1x_3$ $+x_2x_1\bar{x}_3$
		x_2	x_3	x_1	
90	1,3,4,6	1	x_3	x_1	$x_1\bar{x}_3+\bar{x}_1x_3$
95	0,1,2,3,4,6	x_1	x_3	x_1	$x_1\bar{x}_3+\bar{x}_1$
99	0,1,5,6	x_3	x_1	x_2	$x_3\bar{x}_2x_1+x_3x_2$ $\bar{x}_1+\bar{x}_3\bar{x}_2$

TABLE X CONTINUES

Identifier	Function	Input configuration			Algebraic equation
		1	2	3	
101	0,2,5,6	x_3	x_2	x_1	$\bar{x}_3\bar{x}_1+x_3x_1\bar{x}_2$ $+x_3\bar{x}_1x_2$
102	1,2,5,6	1	x_1	x_2	$x_1x_2+x_2x_1$
119	0,1,2,4,5, 6	x_1 x_2	x_2 x_1	x_1 x_2	$x_1\bar{x}_2+\bar{x}_1$
153	0,3,4,7	x_1	0	x_2	$x_1x_2+\bar{x}_1\bar{x}_2$
165	0,2,5,7	x_1	0	x_3	$\bar{x}_1\bar{x}_3+x_1x_3$
170	1,3,5,7	not required			x_1
175	0,1,2,3, 5,7	x_3	x_1	0	$x_3x_1+\bar{x}_3$
187	0,1,3,4, 5,7,	x_2	x_1	0	$x_2x_1+\bar{x}_2$
195	0,1,6,7	x_3	0	x_2	$x_3x_2+\bar{x}_3\bar{x}_2$
204	2,3,6,7	not required			x_2
207	0,1,2,3, 6,7	x_3	x_2	0	$x_3x_2+\bar{x}_3$
221	0,2,3,4, 6,7	x_1	x_2	0	$x_1x_2+\bar{x}_1$
240	4,5,6,7	not required			x_3
243	0,1,4,5, 6,7	x_2	x_3	0	x_3+x_2

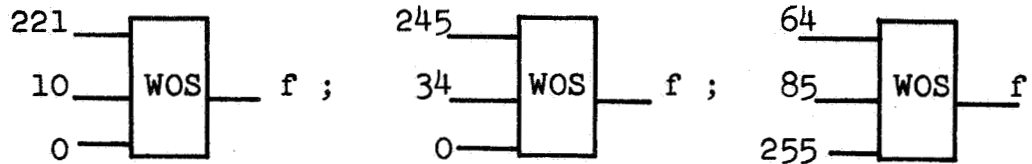
TABLE X CONTINUES

Identifier	Function	Input configuration			Algebraic equation
		1	2	3	
245	0.2.4.5, 6,7	x_1	x_3	0	$\bar{x}_1 + x_3$
255	0,1,2,3, 4,5,6,7	not required			1

As an illustrative example, let us synthesize the function

$$f = x_1 \bar{x}_2 + x_1 \bar{x}_3 .$$

Since the synthesis method is identical to the one used in the previous section, we only show here the results of the computer run (Appendix III). The lowest solution weight found was 8 and three acceptable solutions were obtained :



These three solutions represent the following three circuits (see TABLE X)

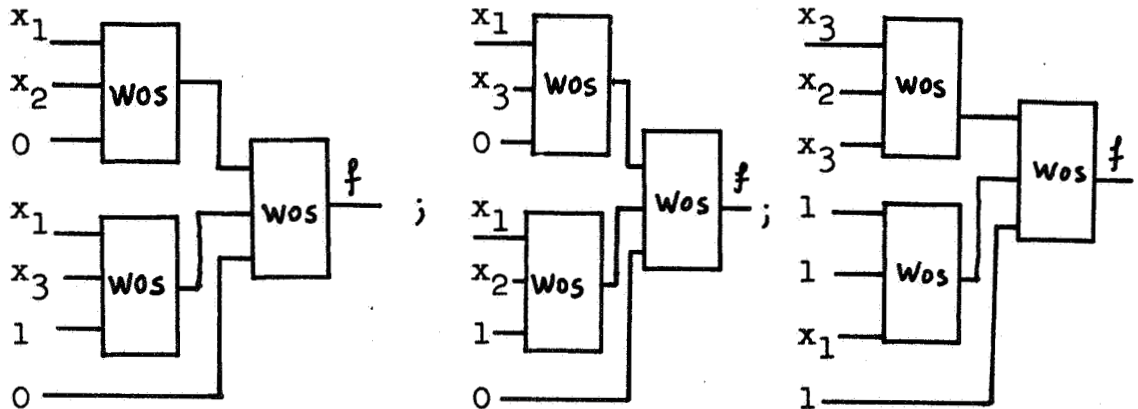
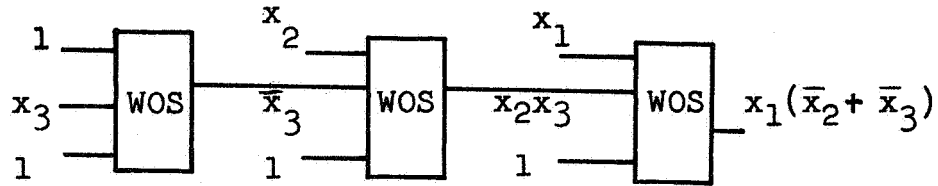


Figure 25
 WOS Synthesis Example.

The circuits of Figure 25 contain two logic levels only. The circuit proposed by Patt in reference 34 is the following



which contains three logic levels.

4.4.2 Synthesis of Multiple-Output Logical Nets

A natural extension of the synthesis algorithm presented in Section 4.4.1 is its application to multiple-output combinational logic nets.

The multiple-output implementation problem is usually connected with some optimization requirement. Minimizations are stated as a goal for minimum number of characters 30, 39, 35, minimum number of logic elements, 30, 39, 40, 35,

minimum number of external pins to the circuit, minimum stages, ³⁵ optimum fan-in, fan-out, etc. The optimization is in close relation with the means of solution in all cases cited above. In our case the means of solution is the BA and a maximum sharing of modules principle is introduced here.

In this section we outline a method for solving the multiple-output synthesis problem using the Boolean Analyzer. This method is based on the frequency v_1 of occurrence of each input function F_1 when solving a system of

Boolean equations.

A multiple-output combinational network has the general structure of Figure 26.

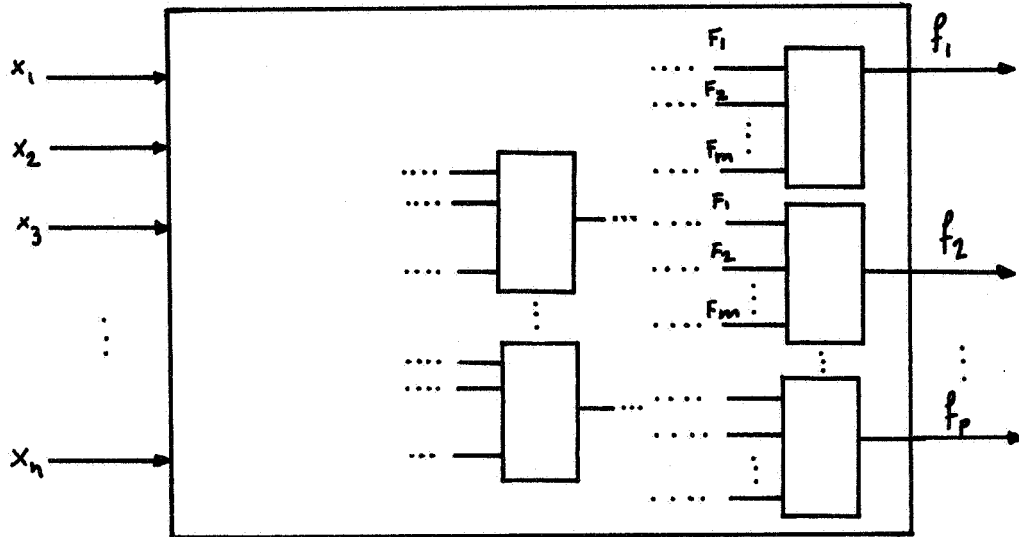


Figure 26

Multiple-Output Combinational Net.

Suppose that the inventory of available modules is restricted to one type, as in the case considered in Sections 4.4.1.4 and 4.4.1.5. The problem of finding the multiple-output circuit with the maximum sharing of modules may be solved in the following way :

Step 1 : For every output function f_j ($j=1,2,\dots,p$) we solve the equation

$$f_j(x_1, x_2, \dots, x_n) = \mathcal{F}(F_1, F_2, \dots, F_m) \quad (40)$$

where \mathcal{F} is the function realized by the module of the inventory, and F_1, F_2, \dots, F_m the unknown variables.

Step 2 : For every function f_j the set S_j ($j = 1, 2, \dots, p$) of solutions of Eqn. (40) with lowest weight is computed. (Steps 1 and 2 constitute the algorithm for the single-output case considered in Section 4.4.1.2).

Step 3 : Every possible combination $S = \{S_1, S_2, \dots, S_j, \dots, S_p\}$ of the solutions obtained in Step 2 is considered and a weight factor W is computed taking in consideration the frequency v_i of occurrence of every input function F_i ($i = 1, 2, \dots, m$) in S and the weight-factor w_i assign to every function F_i :

$$W = \sum_{j=1}^p \left[\sum_{i=1}^m v_i w_i \right]_j . \quad (41)$$

Step 4 : A combination S with largest weight-factor W represents an implementation with maximum sharing of modules.

To illustrate the above described synthesis process following is an example of multiple-output WOS-circuits synthesis.

Example : Synthesize the following functions

$$f_1 = x_1 \bar{x}_2 + x_1 \bar{x}_3$$

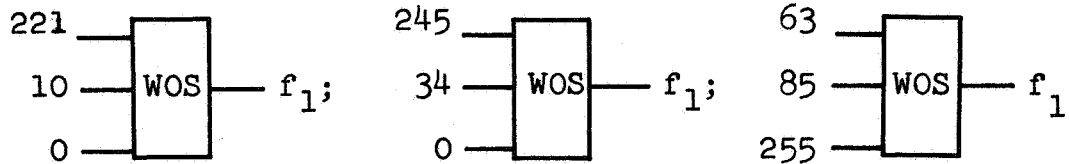
$$f_2 = x_1 x_2 x_3$$

using maximum sharing of modules and assuming the inventory functions corresponding to WOS-modules of TABLE X.

Steps 1 and 2 : We solve the equation

$$x_1 \bar{x}_2 + x_1 \bar{x}_3 = F_1 \bar{F}_3 F_2 + F_1 F_3 \bar{F}_2 + \bar{F}_1 \bar{F}_3$$

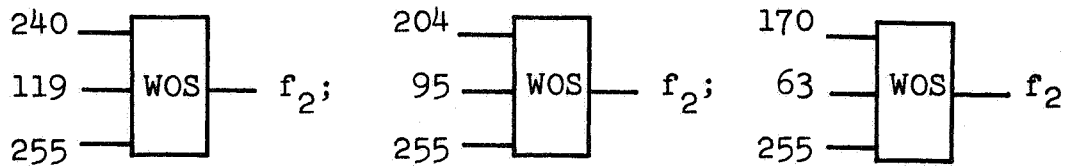
using the SBEV4 program of Appendix III . Three solutions were found with weight equal to 8 :



Similarly, we solve equation

$$x_1 x_2 x_3 = F_1 \bar{F}_3 F_2 + F_1 F_3 \bar{F}_2 + \bar{F}_1 \bar{F}_3$$

Three solutions with lowest weight (in this case equal to 5) were found:



Step 3 : We consider now all possible combinations of solutions obtained in Step 2 . We assume here that weight factors ω_i are assigned to input functions F_i following the criterium used in Section 4.4.1.3.

Nine combinations of solutions are possible,

Combination	f ₁			f ₂			weight-factor W Eqn. (41)
	F ₁	F ₂	F ₃	F ₁	F ₂	F ₃	
1	221	10	0	240	119	255	13
2	221	10	0	204	95	225	13
3	221	10	0	170	63	255	13
4	245	34	0	240	119	255	13
5	245	34	0	204	95	255	13
6	245	34	0	170	63	255	13
7	63	85	255	240	119	255	13
8	63	85	255	204	95	255	13
9	63	85	255	170	63	255	21

TABLE XI

MAXIMUM-SHARING COMPUTATION

and for each one of them a weight factor W is computed according to Eqn. (40) (see TABLE XI).

Step 4 : A circuit with maximum sharing of modules is obtained by choosing the combinations of solutions with largest weight-factor W . In this case there is only one combination with $W = 21$ (TABLE XI). This combination corresponds to the following circuit :

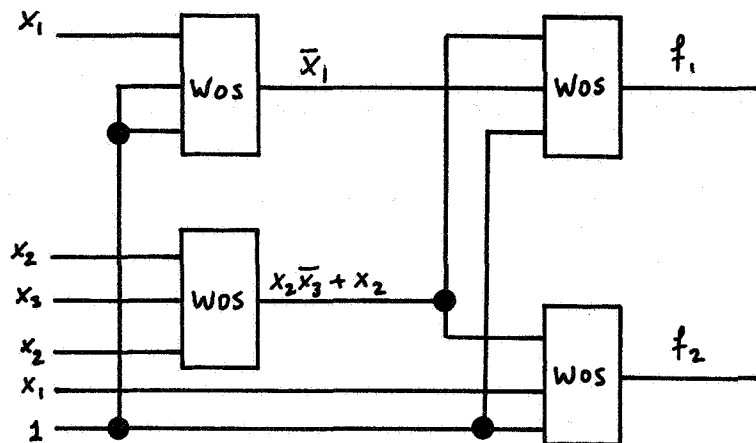


Figure 27. Multiple-Output WOS-Synthesis Example.

The automatization of the above multiple-output synthesis procedure can be done using the Boolean Analyser. For Steps 1 and 2 the same method as the one used in the single-output case may be used. The Boolean Analyzer computes the discriminant of the systems of Boolean equations to be solved. For steps 3 and 4 additional programming should be done to compute the weight factors W of each possible solution combination. A further development of the BA system including a multiple-access RAD storage system could assist in the efficient solution of this part of the problem which looks rather lengthy for problems with large number of variables.

CHAPTER V

COMPUTER SIMULATION OF THE B.A.SYSTEM

In Chapter III the hardware structure of the Boolean Analyzer system was presented and in Chapter IV some applications were investigated. At the present time the BA is not constructed so that it is not possible to profit from its parallel processing capabilities when used in a particular application. However, it is possible to simulate the basic modes of operation of the BA by means of computer programming and thus obtain a working software package which could be used in the applications presented in the previous chapter. The principal characteristic of the BA unit, namely its parallel processing, cannot be obtained with computer programs executed in sequential machines and therefore our B.A. simulator will require far more computer-time than the BA hardware unit.

The purpose of this chapter is to present a working BA simulator and to show experimental results.

5.1 Simulation of the modes of operation of the BA

The BA Simulator (BAS) (Appendix IV) consists of a FORTRAN IV program and three subprograms written in SDS SIGMA 7 Assambly Language.

The FORTRAN IV program is composed of the following

parts:

1) A main program

2) Five Subroutines:

Subroutine BASE computes the base three equivalent of a decimal integer number.

Subroutine BINARY computes the base two equivalent of a decimal integer number.

Subroutine TRACK reads in or writes on RAD a bit corresponding to a cancellation process following the description of Section 3.3. This subroutine uses two special functions, IOR and IAND*, which are programmed in assembly language.

Subroutine DISCRI retrieves bit-by-bit from RAD the discriminant of a given system of equations or if desired, may be used to retrieve the bits of a given function. This subroutine uses an assembly language subprogram called BIN** to directly address and retrieve a bit of a word in RAD.

Subroutine MASK determines the cancellation of the implicants of a certain prime-implicant. This subroutine is a direct implementation of Theorem 3, Section 3.1.

*These functions were programmed by Jean L. Baer.

**This subroutine was programmed with the collaboration of Patricia Rubins.

The different subsystems of the BA hardware unit, as presented in Section 3.2, have been simulated reflecting the hardware limitations of the proposed model. Thus, the Logic Operation Unit (LOU) is simulated by an array named LOU with a working dimension of 100x22 (i.e. 100 registers allocating a maximum of 100 Boolean terms of up to a maximum of 22 variables) and the circulating memory, RAD, is simulated by an array named RAD of dimension 10000. These dimensions may be extended up to the limit allowed by the memory size of the machine in use. In our case a SDS Sigma 7 computer was chosen to run the simulation programs because it is the general purpose computer proposed to work in conjunction with the BA hardware unit (Fig. 7).

5.1.1 Simulation of MODE I

The flowchart corresponding to MODE I was presented in Figure 9, Section 3.3.1 and its corresponding FORTRAN IV program is shown in Appendix IV.

The interesting part of this program corresponding to the implementation of the NON-IMPLICANT CANCELLATION Theorem (Theorem I, Section 3.1) is the following:

```
      ⋮  
      N1= 3**N  
      DO 110 I=1,N1  
      L=I-1
```

```

CALL BASE (L,TC)
DO 108 K=1,NT
DO 104 J=1,22
TEMP LOU(K,J) TC(J)
IF(TEMP.EQ.3) GOTO 108
104 CONTINUE
R=1
L=I-1
CALL TRACK (L,A,R)
GOTO 110
108 CONTINUE
110 CONTINUE
.
.
.

```

where every term of $Y=0$ allocated in array LOU is compared with the content of the triadic counter, TC, by addition of corresponding digits. If no addition equals to three for every term of LOU (i.e. for every term of $Y=0$), then the argument R is set to 1 and is entered in Subroutine TRACK which writes a 1 in the corresponding bit position of a word in array RAD. (This writing operation corresponds to a CANCELLATION. See TABLE II). If an addition equals to three, then the next term in array LOU is considered. If all terms in LOU produce at least one addition digit equal to three, then the content of bit position I in array RAD (I equals

✓

to the decimal equivalent L of TC plus one) is left equal to zero.

The above procedure is repeated for the $N_1=3^{**}N$ possible contents of the triadic counter TC .

If more than 100 terms should be processed (this is the case when $Y=0$ contains more than 100 terms) then the above described procedure is carried through for every set of 100 terms leaving, naturally, the array RAD unchanged at the end of every 100-terms-processing operation. This can be done because the cancellation operation is Boolean additive.

The simulation of $MODE I$ in binary operation is done by an almost alike part of program as used for triadic operation, the only difference being that $N_1=2^{**}(N+M)$ (N, M are the number known and unknown variables respectively) and that the triadic counter is replaced by the binary counter (Subroutine $BINARY$).

In both binary and triadic operation of $MODE I$ the string of bits of array RAD represents the space T_c of implicants (TABLE II).

5.1.2 Simulation of $MODE II$

The flowchart corresponding to the algorithmic steps of $MODE II$ was presented in Figure 10, Section 3.3.2. The FORTRAN IV program containing the simulation of this mode of operation of the BA is presented in Appendix IV.

The partial program implementing this mode of operation is the following :

```

      .
      .
      .
113 K=1
      DO 114 K3=1,22
      TC(K3)=0
114 LOU(K,K3)=2
      K=0
      I=1
115 R=0
      L=I-1
      CALL MASK (A,K)
      IF(A.NE.O) GOTO 128
120 L=I-1
      CALL BASE (L)
121 K=K+1
      DO 122 J=1,22
122 LOU(K,J)=TC(J)
      L=I
      CALL BASE (L)
      CALL MASK (A,K)
124 I=I+1
128 IF(A) 129,130,129
129 I=I+1
      L=I-1
      CALL BASE (L)

```

```
130 IF(I.GT.N1) GOTO 115
```

```
145 WRITE (6,1006)
```

....

which basicly represents the implementation of Theorems 2& 3 (Section 3.1) as follows:

The bits of array RAD are orderly read out (argument R 0 is used for the reading operation in Subroutine TRACK) according to their identifier I . The first non-zero bit of RAD encountered corresponds to a prime implicant (Theorem 2 Section 3.1). The triadic equivalent(content of TC) of its identifier L (L=I-1) is stored in LOU. At this stage I is step up by one unit and Subroutine BASE is called to compute the triadic equivalent of L=I-1. The content of TC is now compared digit-by-digit with all the terms in LOU which have been selected already as prime implicants.(This is the task of Subroutine MASK which is a direct implementation of Theorem 3,Section 3.1). If the out coming parameter A from Subroutine MASK equals to zero, then the term TC under consideration is a prime implicant if the corresponding bit in RAD is also a zero(meaning that TC is an implicant of the given function $y \bar{Y}$). Only under these conditions the content of TC is considered as a new primr implicant and stored in the corresponding address K of LOU. The above procedure continues until the identifier I exceeds $N1=3**N$ (N being the number of variables of the given function). For problems with higher number of prime implicants than the capacity

of array LOU the following procedure is used : At the moment the last prime implicant has completed the array LOU a cancellation of all implicants of the terms in LOU takes place in RAD. After this cancellation has been completed the terms in LOU are stored on magnetic tape or disc and the procedure (Mode II operation) starts again with I equal to the identifier corresponding to the last prime implicant obtained. When $I=3**N$ the set of prime implicants are retrieved from tape or disc.

5.2 Experimental results

The BAS was used in two different experiments:
1) in the determination of the discriminant of large systems of Boolean equations, and 2) in the determination of prime implicants of functions with large number of variables.

The objective of these experiments was to obtain BAS execution time measurements and to compare them with the estimated performance of the BA hardware unit ¹⁸. The results of the experimental runs are reported in Figures 28 and 29.

In figure 28 the computation time in the discriminant determination versus the number of input terms of the given equation $Y=0$ is plotted. Functions of 5, 10 and 15 variables were tested. Similarly, Figure 29 reports the results obtained in the determination of prime implicants.

SYSTEM OF BOOLEAN EQUATIONS
DISCRIMINANT DETERMINATION

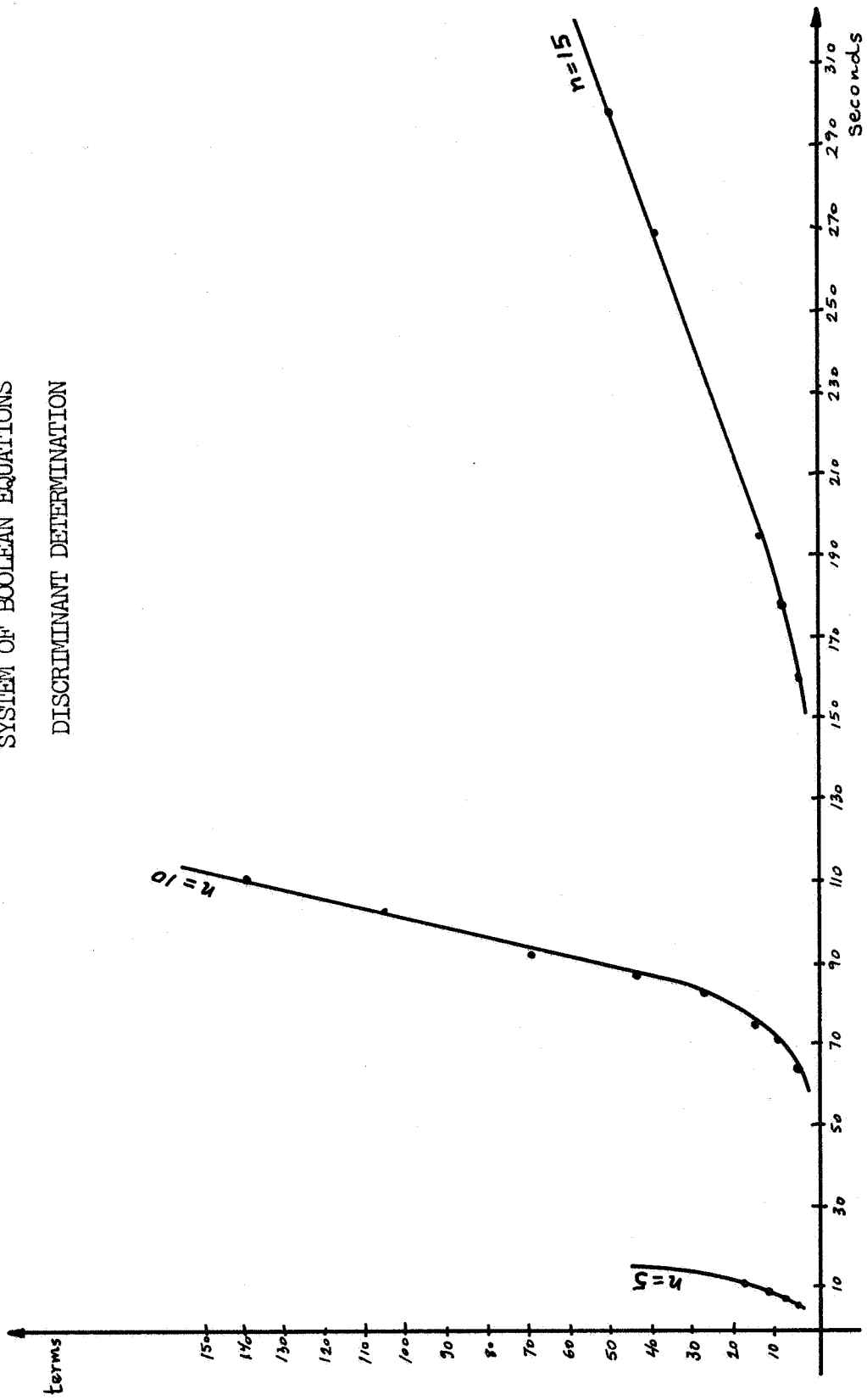


Figure 28. System of Boolean Equations Discriminant.

PRIME IMPLICANTS DETERMINATION

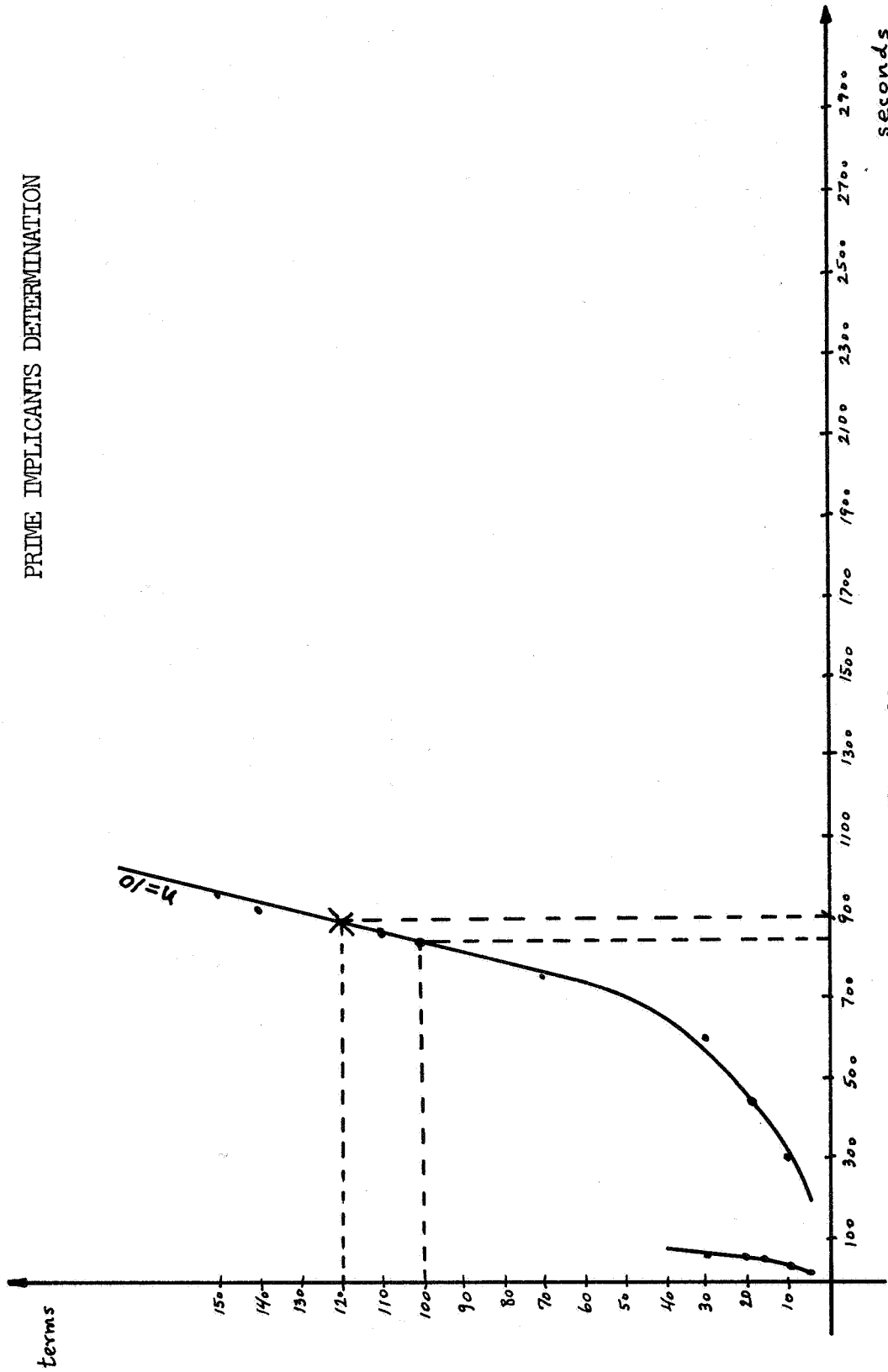


Figure 29 . Prime Implicants Determination.

Functions of 5 and 10 variables were tested containing up to 150 terms each.

From the data presented in Figures 28 and 29 a rough comparison with the BA hardware unit estimated performance may be obtained in the following way :

Assume $Y=0$ with $n=10$ variables and 120 terms.

1) Discriminant determination :

Estimated execution time of BA hardware unit¹⁸:

.005 seconds

Execution time of BA simulator :

108 seconds

Ratio : 21×10^3

2) Prime-implicant determination :

Estimated execution time of BA hardware unit¹⁸:

.06 seconds

Execution time of BA simulator :

900 seconds

Ratio : 15×10^3 .

CHAPTER VI

CONCLUSIONS

In this dissertation we have shown how a new operation unit, called Boolean Analyzer¹⁸, is integrated with a general purpose computer (SDS Sigma 7 computer) to solve certain kind of problems encountered in logic design.

After a brief survey of methods for solving systems of Boolean equations (Chapter I) the Boolean Analyzer was presented as a part of the UCLA Variable Structure Computer System (Chapter II) and a computer organization of the system formed by Sigma 7 and the Boolean Analyzer unit was proposed. It was shown that an improvement of a certain power of 3 could be achieved depending upon the number of Boolean Analyzer (BA) processors working in parallel.

In Chapter III we briefly restated, and for the seek of completeness, the basic theorems on which the parallel processing feature of the BA is based. Its two modes of operation were explained in detail.

The applications of the BA to solve certain problems in logic design were investigated in Chapter V: (Table XII)

- 1) A procedure for solving large systems of Boolean equations was given and a computer program, called SOLDET, Appendix I, was developed to obtain all solutions of a given system.

- 2) A method was proposed to solve the classical problem of irredundant coverings of a given Boolean function

TABLE XII. APPLICATIONS OF THE BOOLEAN ANALYZER

Application Description	B.A. Operation Mode	General Purpose Computer Task (Sigma 7)	Simulation Programs available
Prime implicants determination	Mode I (triadic) + Mode II.	Provides control, retrieves list of prime implicants.	Boolean Analyzer Simulator (BAS) in Mode 1.
Discriminant computation	Mode I (binary)	Provides control, retrieves discriminant	BAS in Mode 2
Systems of Boolean Eqns.	Mode I (binary)	Solutions computed by subroutine: SOLDET	SBEV4 in Mode 1.
Irredundant coverings of Boolean Function y.	Mode I (triadic) + Mode II. (P.I. of y) Mode I + Mode II (P.I. of z)	Computer Function Z Retrieves P.I. of z.	
TANT Synthesis of Boolean Function y.	Mode I (triadic) + Mode II (P.I. of y) Mode I (binary) (G.P.I. of y) Mode I (triadic) + Mode II. (P.I. of z)	Provides control, selects proper solutions from discriminant. Computes Z. Retrieves P.I. of z.	BAS (Mode 1) + SBEV 3
Synthesis of combinational nets from fixed inventory of modules	S.O.	Proper selection of acceptable solutions. Iterations if required	SBEV4 (Mode 2 or 3)
	M.O.	Proper combination of acceptable solutions. Maximum sharing computation. Iterations if required.	SBEV4 + additional programming required.

LEGEND: B.A. = Boolean Analyzer; P.I. = Prime Implicant; G.P.I. = Generalized Prime Implicant; S.O. = Single-Output; M.O. = Multiple-Output.

by properly using the two modes of operation of the BA. The main characteristic of this approach is the ordered processing of Boolean terms which eliminates the need of combinatorial comparisons between terms and makes possible the representation of a Boolean term by only one bit of information.

3) The TANT synthesis problem of logic nets was studied and it was found that it may be solved by computing the so called generalized prime implicants of the given function which are the solutions of a certain system of Boolean equations. A computer program, called SBEV3, Appendix II, was developed to obtain all the generalized prime implicants of functions up to 3 variables. (This limitation may be removed by using the BA Simulator presented in Chapter V). It was found that the results obtained agreed with those computed manually.

4) A new approach to the general synthesis problem of combinational logic networks using a fixed inventory of integrated circuit modules was developed. The proposed method consists of solving a system of Boolean equations at every logic level of the network to obtain all possible input functions. The number of possible solutions may be reduced automatically by adding designer's constraints to the inventory of available functions. This inventory is part of the input data. The reduced number of acceptable solutions that match the inventory are classified according to a

weight factor which again may be specified by the designer. The solutions with the desirable weight factor are retrieved as optimum. A FORTRAN IV computer program called SBEV4 implementing the proposed synthesis method was developed for problems involving up to 9 variables*. This program can work in three different modes: Mode I generates all existing solutions, Mode II generates all properly weighted solutions, and Mode III generates the solutions with the lowest weight. The selection of mode of operation is done in the first data card. To prove that the proposed synthesis algorithm is independent of the inventory of modules the synthesis program was used to synthesize circuits using NAND logic elements and WOS modules. In the first case the circuit obtained manually by standard techniques was found to be included in the set of solutions given by the program. In the second case circuits were obtained that contain less logic levels than their equivalent given in the literature.

4) An extension of the single-output synthesis method to the multiple-output case was investigated. A new objective in synthesis was sought: the maximum sharing of modules at the last logic level. An algorithm was developed which uses as a first step the single-output synthesis method previously studied. The acceptable solutions to a logic level corresponding to every output function are combined

*This limitation may be removed by using a memory array for every term instead of a single word per term as done in the SBEV4 program.

in all possible ways and a weight factor is computed to determine which combination of solutions produces the maximum sharing of modules. An example of multiple-output WOS-modules synthesis is presented and manually solved. Further programming effort is needed in order to automat the multiple-output synthesis algorithm.

Finally a Boolean Analyzer Simulator (BAS) was developed (Chapter V). Its motivation was two fold: 1) To provide a working software package that simulates the modes of operation of the BA in order to obtain rough computing time measurements which could be compared with the estimated processing time of the Boolean Analyzer hardware unit¹⁸. The execution time ratio between the BA unit and the BAS was found to be of the order of 10 to 20 thousand. The BAS program is limited to 22 variables as the proposed BA hardware unit. The number of terms that can be processed has no limitation but the available computing time. This feature is achieved by using peripheral storage media (tape or disc)

2) To provide an experimental tool for investigation of other problems in logic or related areas. The BAS program will prove to be a valuable tool in computing the number of existing solutions of such problems and with the help of subprogram SOLDET(Appendix I) all possible solutions may be obtained if so desired.

The main contribution of this dissertation has been to present a new philosophy in the field of automated

solution of problems encountered in the design of logic circuits. This philosophy is, however, not limited to the kind of problems presented here but we may say that in general any problem that can be stated as a system of Boolean equations may be solved by the automated methods introduced in this work.

Fields for future research, where Boolean methods are being used, are: Synthesis of Threshold Logic, Sequential Circuits Synthesis and the Challenging field of Pseudo-Boolean Functions and their Applications to Operations Research problems.

BIBLIOGRAPHY

1. Boole, George, An Investigation of the Laws of Thought, Dover, New York, 1854.
2. Bernstein, B. A., "Note on the Condition that a Boolean Equation Have a Unique Solution," Amer. J. Math., 54: 417-418, 1932.
3. Goto, M., "On the General Solution of a Logical Equation with many Unknowns," Bull. Electr. Lab., 20:81-87, 1956.
4. Zemanek, H., "Die Lösung von Gleichungen in der Schaltalgebra,:" Arch. Elektr. Übertragung, 12: 35-44, 1958.
5. Rouche, N., "Some Properties of Boolean Equations," IRE Trans. EC-7:291-298, 1958.
6. Phister, Montgomery, Logical Design of Digital Computers, John Wiley, New York, 1958.
7. Shubert, E. L., "Simultaneous Logical Equations," Communic. and Electr., 46:1080-1083, 1960.
8. Klir, J., "Reseni soustav Booleovych rovnic," Aplik. Mat., 7:265-272, 1962.
9. Carvallo, M., "Detail sur la Resolution des Equation Booleennes en vie du Codage," Publ. Institut Statistique Paris, 13:21-44, 1964.
10. Rudeanu, S., "Boolean Equations and Their Applications to the Study of Bridge-Circuits. I," Bull. Math. Soc. Sci. Math. Phys. de la R. P. Roumaine, 3, 51:445-473, 1959.
11. Campeau, Joseph O., The Synthesis and Analysis of Counters in Digital Systems by Boolean Matrices, M.S. in Engineering, University of California, Los Angeles, 1955.

12. Hohn, F. and R. Schissler, "Boolean Matrices and the Design of Combinational Relay Switching Circuits," Bell Technical Journal, 34:177-202, January 1955.
13. Ledley, Robert S., Digital Computers and Control Engineering, McGraw-Hill, New York, 1960.
14. Ledley, R. S., "Boolean Matrix Equations in Digital Circuit Design," IRE Trans., EC-8:131-139, 1960.
15. Maitra, K. K., "A Map Approach to the Solution of A Class of Boolean Equations," Commun. and Electron. 59:34-36, 1962.
16. Svoboda, A. "An Algorithm for Solving Boolean Equations," IEEE Trans. on Electron. Computers, EC-12:557-558, October 1963.
17. Marín, M. A., "A FORTRAN IV Program for Solving Boolean Equations," University of California, Los Angeles, Digital Technology Research Group, Internal Memorandum 64, December 1967.
18. Svoboda, A., "Boolean Analyzer" (To be published in Proc. of IFIP Conference, 1968).
19. Estrin, G., "Organization of Computer Systems: The Fixed Plus Variable Structure Computer," Proceedings of the Western Computer Conference, San Francisco, California, May 3-5, 1960.
20. Svoboda, A., "Ordering of Implicants," IEEE Trans. on Elec. Computers, EC-16:100-105, February 1967.
21. Laster, Lonnie, The Logical Design of the Boolean Analyzer, M. S. in Engineering, University of California, Los Angeles, 1967.
22. Gilley, George, Design of the Interface Between SDS Sigma 7 Computer and the UCLA Boolean Analyzer, M.S. in Engineering, University of California, Los Angeles, 1967.
23. Svoboda, A., "Irredundant Coverings," University of California, Digital Technology Research Group, Internal Memorandum December 1966.
24. Marley, G. A. and S. Orgen, "Minimal Three-Level NAND Circuitry," Data Systems Division, IBM Corp., TROO.933, November 1962.

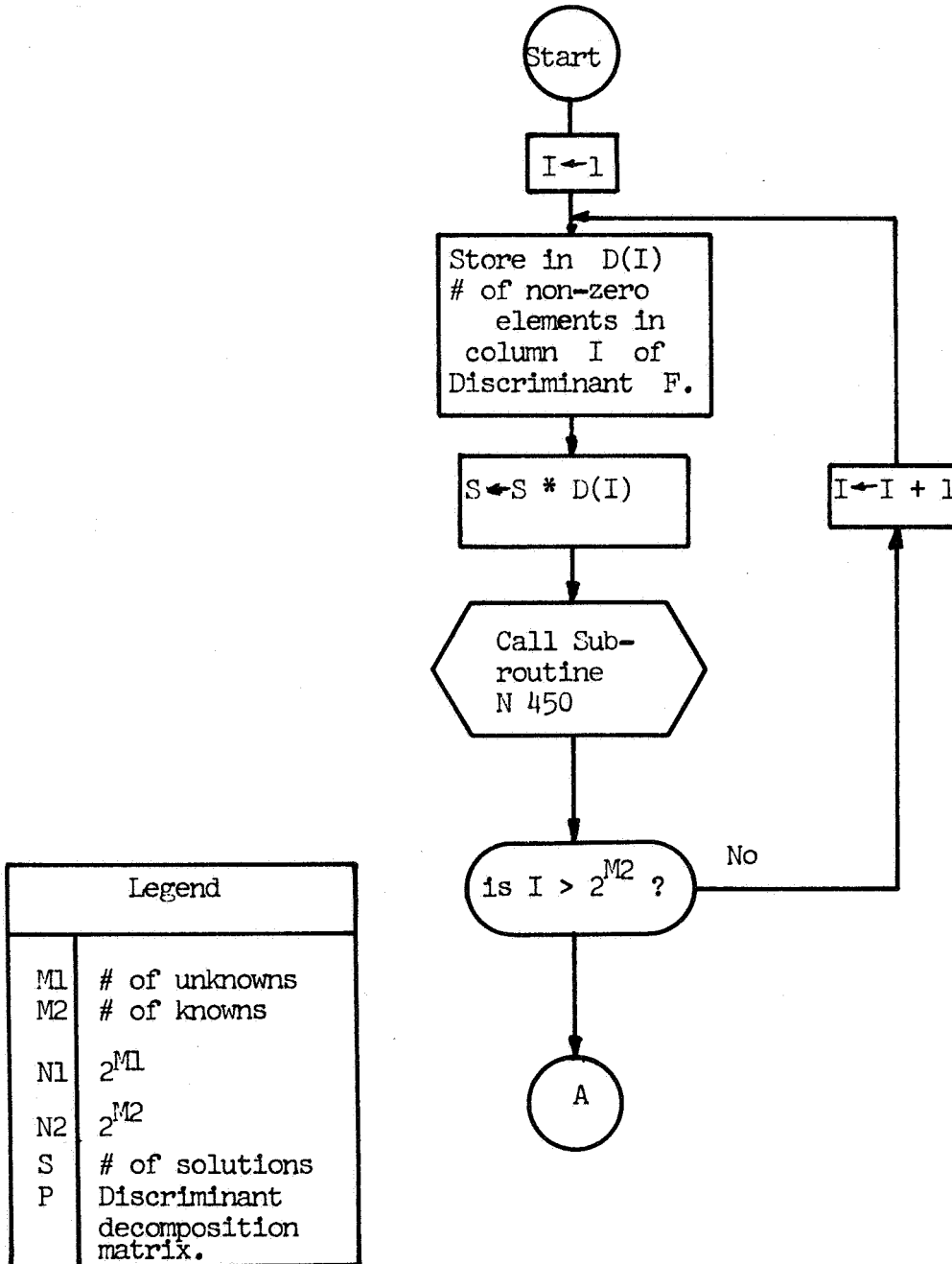
25. McCluskey, E. J., "Logical Design Theory of NOR Gate Networks with no Complemented INputs," Proceedings of the 4th Annual Symposium on Switching Circuit Theory and Logical Design, Chicago, Illinois, pp.137-148, October 1963.
26. Hellerman, L., "A Catalog of Three-Variable OR INVERT and AND-INVERT Logical Circuits," IEEE Trans. on Electron. Computers, EC-12; 198-223, June 1963.
27. Marley, Gerald A. and John Earle, The Logical Design of Transistor Digital Computers, Prentice Hall, 1963.
28. Gimpel, J. F., "The Minimization of TANT Networks," IEEE Trans. on Electronic Computers, EC-16:18-38, February 1967.
29. Petrick, S. R., "A Direct Determination of the Irredundant Forms of a Boolean Function from the Set of Prime Implicants," Mass. Tech. Rept. No. AFCRC-TR-56-110, 1956.
30. McCluskey, Edward J., Introduction to the Theory of Switching Circuits, McGraw-Hill, New York, 1965.
31. Luccio, F. and A. Grasselli, "A Method for Minimization of the Number of Internal States in Incompletely Specified Sequential Networks," IEEE Trans. on Electronic Computers, EC-14:350-359, June 1965.
32. Bartee, Thomas C., Irwin L. Lebow and Irving S. Reed, Theory and Design of Digital Machines, McGraw-Hill, New York, 1962.
33. Patt, Y. N., "Synthesis of Switching Functions Using Complex Logic Modules," 1967 Spring Joint Computer Conference, AFIPS Conference Proceedings, 30:699-706, 1967.
34. Patt, Y. N., "Synthesis of Switching Functions Using a Minimum Number of Integrated-Circuit Modules," Stanford Electronics Laboratory, Technical Report No. AFAL-TR-67-142, December 1966.
35. Schneider, P. R. and D. L. Dietmeyer, "An Algorithm for Synthesis of Multiple-Output Combinational Logic," IEEE Trans on Computers, EC-17:117-128, February 1968.

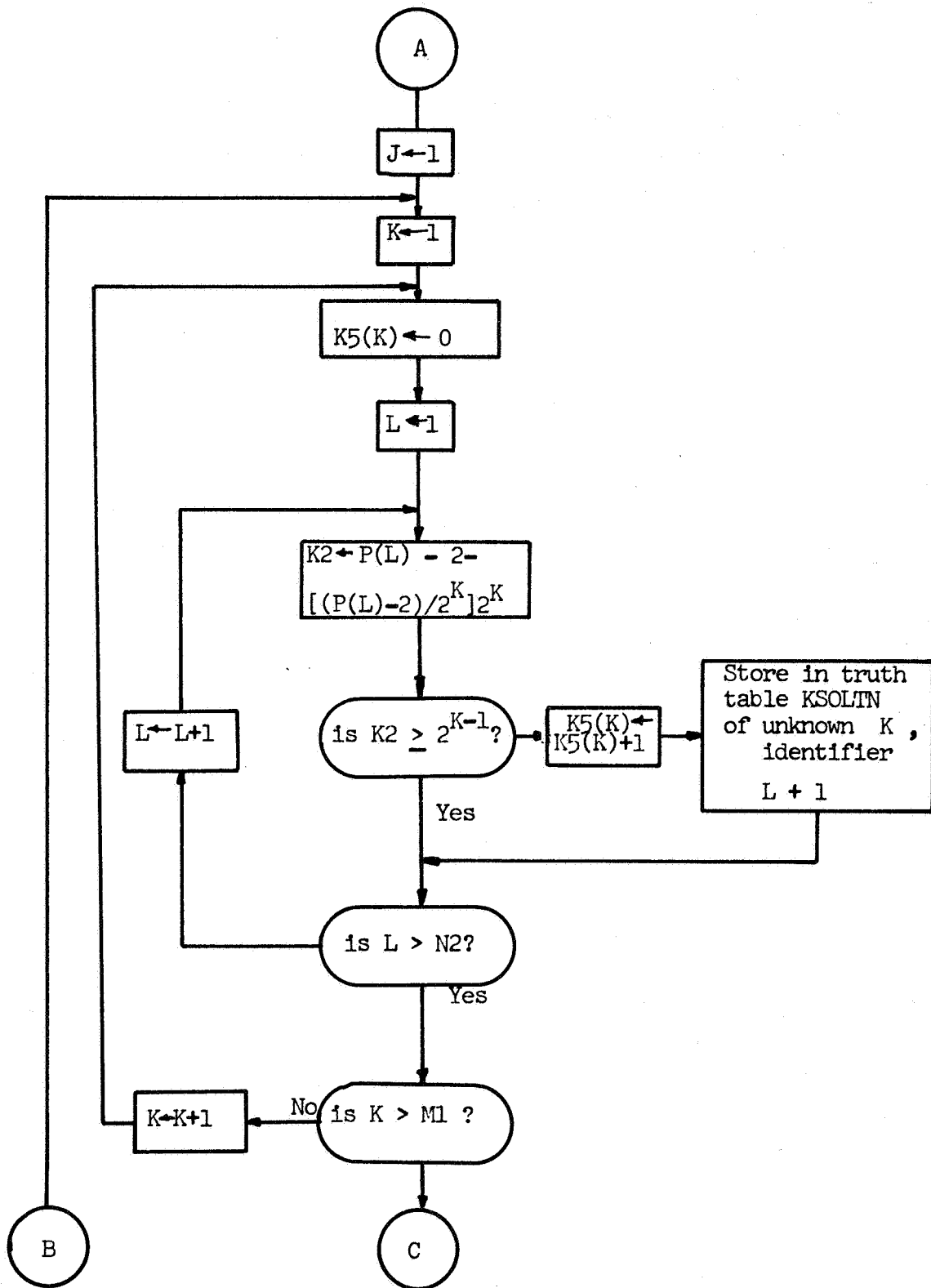
36. Ashenurst, R. L., "The Decomposition of Switching Functions," Pro. International Symposium Theory of Switching (April 2-5, 1957), Vol. 29 of Annals of Computation Laboratory of Harvard University, Cambridge, Mass., 1959, pp. 74-116.
37. Roth, J. P., "Minimization over Boolean Trees," IBM Journal of Research and Development, 4:543-558, November 1960.
38. Marín, M. A., and M. A. Melkanoff, "Canonical and Minimal Forms for NAND and NOR Logics, University of California, Los Angeles, Department of Engineering, Report No. 65-57, December 1965.
39. Bartee, T. C., "Computer Design of Multiple-Output Logical Networks," IEEE Trans. on Electronic Computers, EC-10:21-31, March 1961.
40. Marín, M. A., "Minimizacion por Extension de Circuitos Combinacionales de Salidas Multiples," Revista de la Real Academia de Ciencias Exactas, Fisicas y Naturales, 61 :175-188, Madrid 1967.

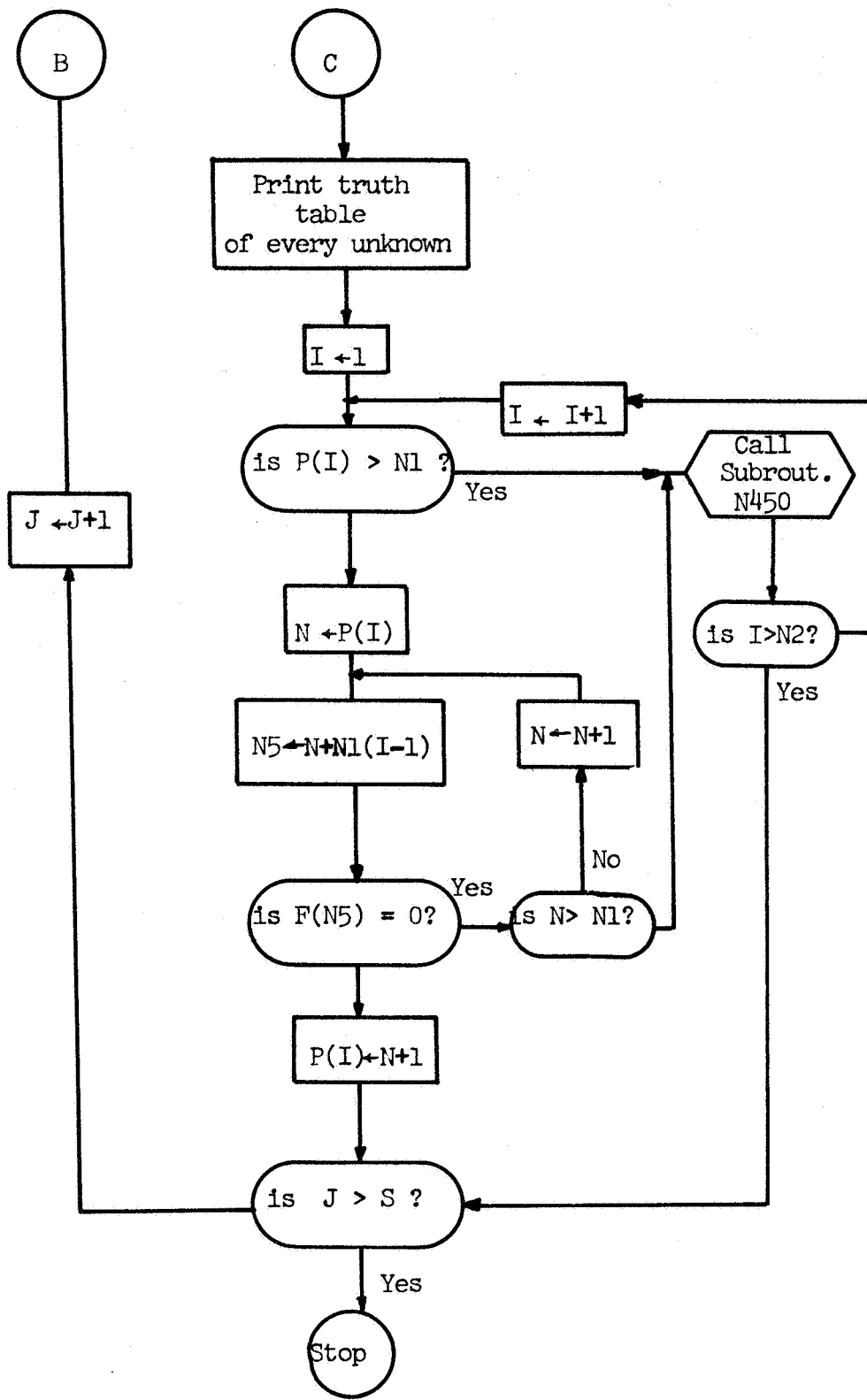
APPENDIX I

PROGRAM SOLDET

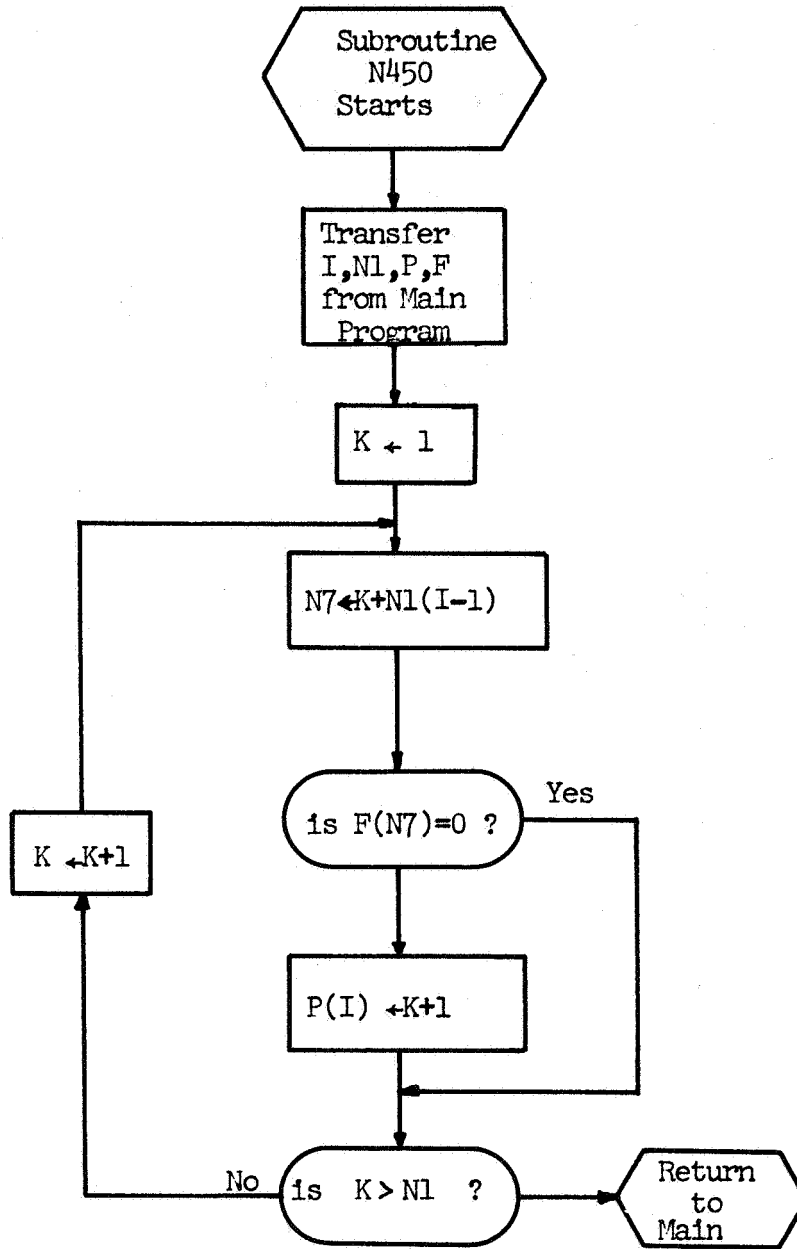
A. Flowchart corresponding to the program, called SOLDET, which computes the truth tables of the unknown variables for every solution of the given system of Boolean equations.







B. Flowchart corresponding to Subroutine N450.



C. Listing of Program
SOLDET

```

555 S=1
      DO 40 I=1,N2
      S=S*D(I)
      CALL N450 (N1,I)
40 CONTINUE
44 DO 80 J=1,S
      DO 69 K=1,M1
      DO 5 J10=1,16
      5 KSOLTN (J10,K)=0
      K5(K)=0
66 DO 70 K1=1,N2
      K3=K1-1
      K2=P(K1)-2-((P(K1)-2)/2**K)*2**K
      IF(K2-2**(K-1)) 70,68,68
68 K5(K)=K5(K)+1
      KLL=K5(K)
      KSOLTN(KLL,K)=K3+1
70 CONTINUE
69 CONTINUE
65 DO 79 K=1,M1
      IF (K-1) 46,64,46
46 IF (K-2) 48,62,48
48 IF (K-3) 50,60,50
50 IF (K-4) 52,58,52
52 IF (K-5) 54,56,54
54 WRITE (108,206)
      WRITE (6,1900) IDENT(K)
206 FORMAT (3H F=)
      GO TO 75
56 WRITE (108,207)
207 FORMAT (3H E=)
      WRITE (6,1900) IDENT(K)
      GO TO 75
58 WRITE (108,208)
208 FORMAT (3H D=)
      WRITE (6,1900) IDENT(K)
      GO TO 75
60 WRITE (108,209)
209 FORMAT (3H C=)
      WRITE (6,1900) IDENT(K)
1900 FORMAT (' IDENTIFIER:',I10)
      GO TO 75
62 WRITE (108,220)
220 FORMAT (3H B=)
      WRITE (6,1900) IDENT(K)
      GO TO 75
64 WRITE (108,204)
204 FORMAT (1H0,19H 'SOLUTIONS NUMBER')
      IF(MODE.NE.1) GO TO 560
      WRITE(108,205) J

```

```

61 WRITE (6,210)
210 FORMAT (3H A=)
    KLP=K5(K)
73 WRITE (6,1002) (KSOLTM(K6,K),K6=1,KLP)
1002 FORMAT(I10)
79 CONTINUE
71 DO 78 I=1,N2
    N8=P(I)
    IF(N8.GT.N1)GO TO 76
    DO 74 N=N8,N1

    N5=N+N1*I-N1
    IF(P(N5).EQ.0) GO TO 74
    P(I)=N+1
    GO TO 80
74 CONTINUE
76 CALL N450 (N1,I)
78 CONTINUE
80 CONTINUE

```

```

SUBROUTINE N450 (N1,I)
COMMON /RST/F,P
INTEGER F(1023),P(512)
DO 454 K=1,N1
N7=K+N1*I-N1
IF (F(N7)) 452,454,452
452 P(I)=K+1
GO TO 456
454 CONTINUE
456 RETURN
END

```

APPENDIX II
PROGRAM SBEV3

A. Program Listing

```
C*****SBEV3 IS A FORTRAN IV PROGRAM TO COMPUTE
C THE GENERALIZED PRIME IMPLICANTS OF A
C BOOLEAN FUNCTION OF UP TO 3 VARIABLES
  INTEGER TAILS,D,COUNT,T2,STRIP,TEST
  DIMENSION N(25,25),NT(25),NF(8),LIT(25)
  WRITE (6,500)
  WRITE (6,501)
  READ (5,507) LIT
  READ (5,502) NF
  WRITE (6,502) NF
  1 READ (5,503) NBT
    IF(NBT)2,999,2
  2 WRITE (6,504) NBT
    READ (5,505)NIND,NDEP
    WRITE (6,100) NIND,NDEP
  3 INDEX =0
    READ (5,800) TAILS
    DO 7 L3=1,25
    DO 7 L4=1,25
  7 N(L4,L3) = 0
    MX=0
    IR=1
    LPR=0
    STRIP=1
    I=0
    NV=NIND+NDEP
    MV=2**NV
    MI=2**NIND
    MDEP=2**NDEP
  5 READ (5,101) (NT(KP),KP=1,25)
    DO 6 K1=1,NV
    IF(NT(K1)-9) 6,15,900
  6 N(IR,K1)=NT(K1)
    IR=IR+1
    GO TO 5
  15 NTT=IR-1
```

```

WRITE (6,102)
WRITE (6,507) LIT
WRITE (6,103) ((N(LP,LQ),LQ=1,25),LP=1,NTT)
17 D=1
18 INDEX=0
DO 50 K7=1,NTT
IQ=I
DO 32 K8=1,NV
IK=IQ/2
COUNT=MOD(IQ,2)
IF(COUNT) 310,300,310
300 COUNT=2
310 IQ=IK
L7=NTT-K7+1
L8=Nv-K8+1
TEST=COUNT+N(L7,L8)
IF(TEST-3) 32,30,32
30 INDEX=INDEX+1
GO TO 50
32 CONTINUE
GO TO 54
50 CONTINUE
IF(INDEX-NTT) 54,52,54
52 T2=1
GO TO 56
54 T2=0
56 D=D*T2
IF(D) 60,58,60
58 I=STRIP*MI
STRIP=STRIP+1
LPR=LPR+1
IF(I-2**NV) 17,82,82
82 IF(LPR-MDEP) 86,90,86
60 I=I+1
IF(I-STRIP*MI) 18,70,18
70 IX=STRIP-1
IQ=IX
STRIP=STRIP+1
DO 212 N7=1,TAILS
INDEX=0
DO 210 N8=1,NIND
IK=IQ/2
COUNT=MOD(IQ,2)
IF(COUNT) 410,400,410
400 COUNT=2
410 IQ=IK
IF(COUNT-2) 210,200,210
200 INDEX=INDEX+1
210 CONTINUE
IF(INDEX-NIND) 212,84,212
212 CONTINUE

```

```

WRITE (6,104) IX
MX=MX+1
84 IF (I-NV+1) 17,17,86
86 IF (MX) 1,99,1
900 WRITE (6,106)
100 FORMAT (' NUMBER OF UNKNOWNNS =',I2,' NUMBER OF CONSTANTS =',I2/)
101 FORMAT (25I1)
102 FORMAT (' LIST OF TERMS OF THE FUNCTION Y = 0 '/')
103 FORMAT (/25(I1,2X))
104 FORMAT (// ' IDENTIFIER OF THE SOLUTION',2X,I6)
106 FORMAT (' ERROR IN THE INPUT DATA')
107 FORMAT (' NO SOLUTION. NO ROW OF DESCRIPTOR CONTAINS FULL ONES')
108 FORMAT (' NO SOLUTION. ALL FOUND ARE TRIVIAL')
500 FORMAT (' GENERALIZED PRIME IMPLICANTS CALCULATION '/')
501 FORMAT (' MARQUAND MAP OF THE GIVEN FUNCTION')
502 FORMAT (4I6)
503 FORMAT (I2)
504 FORMAT (' TERM OF THE BASE #',I2/)
505 FORMAT (2I2)
507 FORMAT (25(A1,2X))
800 FORMAT (I2)
90 WRITE (6,107)
GO TO 1
99 WRITE (6,108)
GO TO 1
999 STOP
END

```

B. Output corresponding to Table V, Section 4.3.3.

GENERALIZED PRIME IMPLICANTS CALCULATION

MARQUAND MAP OF THE GIVEN FUNCTION

0 0 1 1
 0 1 1 0

TERM OF THE BASE # 1

NUMBER OF UNKNOWNNS = 3 NUMBER OF CONSTANTS = 3

LIST OF TERMS OF THE FUNCTION Y = 0

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	X	Y	Z
0	2	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	2	0	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	2	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	1	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IDENTIFIER OF THE SOLUTION 2

IDENTIFIER OF THE SOLUTION 3

IDENTIFIER OF THE SOLUTION 6

IDENTIFIER OF THE SOLUTION 7

TERM OF THE BASE # 2

NUMBER OF UNKNOWNNS = 3 NUMBER OF CONSTANTS = 3

LIST OF TERMS OF THE FUNCTION Y = 0

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	X	Y	Z
0	2	2	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	2	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	2	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IDENTIFIER OF THE SOLUTION 1

IDENTIFIER OF THE SOLUTION 3

TERM OF THE BASE # 3

NUMBER OF UNKNOWNNS = 3 NUMBER OF CONSTANTS = 3

LIST OF TERMS OF THE FUNCTION Y = 0

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	X	Y	Z
2	2	2	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	2	0	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	2	2	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	2	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IDENTIFIER OF THE SOLUTION 5

IDENTIFIER OF THE SOLUTION 7
TERM OF THE BASE # 4

NUMER OF UNKNOWNNS = 3
NUMER OF CONSTANTS = 3

LIST OF TERMS OF THE FUNCTION Y = 0

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	X	Y	Z
0	0	1	1	1	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	2	2	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	2	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

IDENTIFIER OF THE SOLUTION 4

IDENTIFIER OF THE SOLUTION 6

APPENDIX III

PROGRAM SBEV4

A. Program Listing

```

C****SBEV4 IS A FORTRAN IV PROGRAM TO SYNTHESIZE
C SINGLE-OUTPUT COMBINATIONAL NETWORKS INVOLVING
C UP TO A TOTAL OF 9 VARIABLES.
C SBEV4 WORKS IN THREE DIFFERENT MODES:
C MODE I: SOLUTION OF BOOLEAN EQUATIONS OF UP TO 9 VARIABLES
C MODE II : SOLUTION OF BOOLEAN EQUATIONS RESTRICTED TO AN
C INPUT INVENTORY OF FUNCTIONS. WEIGHT FACTORS ARE
C ASSIGNED TO EACH COMPUTED SOLUTION.
C MODE III: ONLY THOSE SOLUTIONS WITH LOWEST WEIGHT FACTORS
C ARE PRINTED.
C INTEGER F(1023),P(512),D(512),N4(8),OK(8),INPUT(38),KSOLTN(16,16)
C INTEGER ISING(16),K5(16),IDENT(8),T,S,Y,R,H,Q
C INTEGER SOL,OLD,ANSWER(100,10)
C COMMON KSOLTN,OK,INPUT/RST/P,P
C JPL=0
C OLD=100
C 1 READ (5,1001) MODE
C IF(MODE.EQ.0) STOP
C
C GO TO (2,400,400),MODE
C 400 READ (5,1001) NIF
C WRITE (6,1000)
C IF(NIF.EQ.0) GO TO 2
C WRITE (6,2000)
C 2000 FORMAT (' FUNCTIONS OF THE INVENTORY'//)
C READ (5,1010) (INPUT(J),J=1,NIF)

```

```

DO 900 J=1,NIF
WRITE (6,1100) INPUT(J)
1100 FORMAT(I6)
900 CONTINUE
1010 FORMAT (26I3)
1001 FORMAT(I3)
2 READ ( 105,190) M1,M2
190 FORMAT (2I2)
IF (M1) 4,3,4
3 IF (M2) 4,90,4
4 N1=2**M1
6 N2=2**M2
J1=0
7 READ (105,200) (N4(I),I=1,8)
200 FORMAT (8I9)
DO 8 I=1,8
II=I+J1
P(II)=N4(I)
IF (P(II)-900000000) 8,11,8
8 CONTINUE
10 J1=II
GO TO 7
11 T=II-1
WRITE (6,1000)
WRITE(6,1991)
1991 FORMAT (' SYSTEM OF BOOLEAN EQUATIONS: '/')
WRITE (108,310) ( P(I),I=1,II)
310 FORMAT (8I12)
Y=0
DO 38 I=1,N2
D(I)=0
DO 38 J=1,N1
M=J+I*N1-N1
F(M)=M+1
L=0
R=0
H=M
CALL N300 (M1,M2,H,Y)
NJ=0
DO 32 N=1,T
NF=P(N)
IF (F(M)) 13,38,13
13 IF (NF) 12,27,12
12 IF (NF-222222222) 14,27,14
14 IF (NF-444444444) 16,31,16
16 IF (NJ-1) 18,23,18
18 IF (L-1) 20,32,20
20 CALL N400 (Y,NF,Q)
IF (Q) 22,32,22
22 L=1
GO TO 32
23 IF (R-1) 24,32,24
24 CALL N400 (Y,NF,Q)

```

```

        IF (Q) 26,32,26
26 R=1
   GO TO 32
27 NJ=0
   IF (L-R) 28,30,28
28 F(M)=0
   GO TO 38
30 L=0
   R=0
   GO TO 32
31 NJ=1
32 CONTINUE
34 IF (F(M).EQ.0) GO TO 38
36 D(I)=D(I)+1
38 CONTINUE
   IF(MODE.NE.1) GOTO 555
   WRITE (6,1992)
1992 FORMAT (' DISCRIMINANT: '/')
   WRITE (108,500) (D(I),I=1,N2)
500 FORMAT (2I10)
555 S=1
   DO 40 I=1,N2
   S=S*D(I)
   CALL N450 (N1,I)
40 CONTINUE
   IF(MODE.NE.1) GO TO 557
42 WRITE (6,203) S
203 FORMAT (I10,12H 'SOLUTIONS')
557 IF (S.EQ.0) GO TO 84
   IF(MODE.NE.2) GOTO 44
   WRITE (6,1000)
   WRITE (6,1888)
1888 FORMAT (1H0,'ACCEPTABLE SOLUTIONS'/)
44 DO 80 J=1,S
   DO 69 K=1,M1
   DO 5 J10=1,16
   5 KSOLTN (J10,K)=0
   K5(K)=0
66 DO 70 K1=1,N2
   K3=K1-1
   K2=P(K1)-2-((P(K1)-2)/2**K)*2**K
   IF (K2-2**(K-1)) 70,68,68
68 K5(K)=K5(K)+1
   KLL=K5(K)
   KSOLTN (KLL,K)=K3+1
70 CONTINUE
   ISING(K)=0
   OK(K)=0
   CALL CHECK (K,M1,ISING,K5,NIF,IDENT)
   IF (OK(K).EQ.0) GO TO 71
69 CONTINUE

```

```

IF (MODE.EQ.1) GO TO 65
INDEX=0
C****WEIGHT FACTOR ASSIGNMENT
DO 67 K=1,M1
IF (IDENT(K).EQ.0.OR.IDENT(K).EQ.255) GO TO 67
IF (IDENT(K).EQ.170.OR.IDENT(K).EQ.240.OR.IDENT(K).EQ.204) GO TO 15
INDEX=INDEX+4
GO TO 67
15 INDEX = INDEX+1
67 CONTINUE
IF (MODE.NE.3) GO TO 65
IF (INDEX-OLD) 350,360,71
350 SOL=1
360 DO 352 KPL=1,M1
352 ANSWER(SOL,KPL)=IDENT(KPL)
OLD=INDEX
SOL=SOL+1
GO TO 71
65 DO 79 K=1,M1
IF (K-1) 46,64,46
46 IF (K-2) 48,62,48
48 IF (K-3) 50,60,50
50 IF (K-4) 52,58,52
52 IF (K-5) 54,56,54
54 WRITE (108,206)
206 WRITE (6,1900) IDENT(K)
FORMAT (3H F=)
GO TO 75
56 WRITE (108,207)
207 FORMAT (3H E=)
WRITE (6,1900) IDENT(K)
GO TO 75
58 WRITE (108,208)
208 FORMAT (3H D=)

```

```

        WRITE (6,1900) IDENT(K)
        GO TO 75
    60 WRITE (108,209)
    209 FORMAT (3H C=)
        WRITE (6,1900) IDENT(K)
    1900 FORMAT (' IDENTIFIER:',I10)
        GO TO 75
    62 WRITE (108,220)
    220 FORMAT (3H B=)
        WRITE (6,1900) IDENT(K)
        GO TO 75
    64 WRITE (108,204)
    204 FORMAT (1H0,19H 'SOLUTIONS NUMBER')
        IF(MODE.NE.1) GO TO 560
        WRITE(108,205) J
        GO TO 570
    560 JPL=JPL+1
        WRITE(6,205)JPL
    205 FORMAT (I10)
    570 IF(NIF.EQ.0) GOTO 61
        WRITE (6,1920) INDEX
    1920 FORMAT(' WEIGHT FACTOR EQUAL TO ',I10/)
    61 WRITE(6,210)
    210 FORMAT (3H A=)
        WRITE (6,1900) IDENT(K)
    75 IF(ISING(K).EQ.1) GO TO 77
        KLP=K5(K)
    73 WRITE (6,1002) (KSOLTN(K6,K),K6=1,KLP)
    1002 FORMAT(I10)
        GO TO 79
    77 WRITE (6,1003)
    1003 FORMAT (' IDENTICALLY ZERO')
    79 CONTINUE
C*****SOLUTION DETERMINATION: SOLDET.
    71 DO 78 I=1,N2
        N8=P(I)
        IF(N8.GT.N1)GO TO 76
        DO 74 N=N8,N1
            N5=N+N1*I-N1
            IF(F(N5).EQ.0) GO TO 74
            P(I)=N+1
            GO TO 80
    74 CONTINUE
    76 CALL N450 (N1,I)
    78 CONTINUE
    80 CONTINUE
    1000 FORMAT (' *****'/)
        IF (MODE.NE.3) GO TO 91
        SOL=SOL-1
        WRITE(6,1000)
        WRITE(6,1808) OLD
    1808 FORMAT (1H0,'SOLUTIONS WITH LOWEST WEIGHT: W=',I3/)

```

```

DO 370 KPL=1,SOL
WRITE(6,204)
WRITE(6,205) KPL
WRITE (6,280)
280 FORMAT(1H0,'IDENTIFIER OF INPUTS')
370 WRITE (6,1002) (ANSWER(KPL,KM),KM=1,M1)
GO TO 91
84 WRITE (108,213 )
213 FORMAT (38H INDEPENDENT VARIABLES RELATED THROUGH)
WRITE (108,214)
214 FORMAT (44H F=1, WHERE F IS DEFINED BY THE TRUTH TABLE)
DO 88 K=1,N2
NK=K-1
IF(D(K)) 86,88,86
86 WRITE (108,216) NK
216 FORMAT (I4)
88 CONTINUE
91 NT=T+1
DO 89 I=1,NT
P(I)=0
89 CONTINUE
GO TO 1
90 STOP
END

```

```

SUBROUTINE N400 (Y,NF,Q)
INTEGER Y,Q,W,U,U1,W1
Q=1
W=Y
U=NF
DO 412 L1=1,8
U1=U/10000000
W1=W/10000000
IF (W1) 402,408,402
402 IF (U1) 404,408,404
404 IF (U1-W1) 406,408,406
406 GO TO 414
408 U=10*(U-U1*10000000)
410 W=10*(W-W1*10000000)
412 CONTINUE
GO TO 416
414 Q=0
416 RETURN
END

```

```

SUBROUTINE CHECK (K,M1,ISING,K5,NIF,IDENT)
COMMON KSOLTN,OK,INPUT
INTEGER OK(8),INPUT(38),KSOLTN(16,16),ISING(16),K5(16),IDENT(8)
IDENT(K)=0
IF (KSOLTN(1,K).EQ.0) GO TO 510
KLP=K5(K)
DO 505 J7=1,KLP
K9=KSOLTN(J7,K)-1
505 IDENT(K)=IDENT(K)+2**K9
IF(NIF.EQ.0) GO TO 508
DO 507 J8=1,NIF
IF (IDENT(K).EQ.INPUT(J8)) GO TO 508
507 CONTINUE
OK(K)=0
RETURN
508 OK(K)=1
RETURN
510 ISING(K)=1
OK(K)=1
RETURN
END

SUBROUTINE N450 (N1,I)
COMMON /RST/F,P
INTEGER F(1023),P(512)
DO 454 K=1,N1
N7=K+N1*I-N1
IF (F(N7)) 452,454,452
452 P(I)=K+1
GO TO 456
454 CONTINUE
456 RETURN
END

SUBROUTINE N300 (M1,M2,H,Y)
INTEGER H,Y
Y=0
NZ=M1+M2
DO 308 LL=1,NZ
N6=NZ-LL
IF (H-2**N6) 306,306,301
301 H=H-2**N6
Y=10*Y+1
GO TO 308
306 Y=10*Y+7
308 CONTINUE
RETURN
END

```


B. Output corresponding to the Synthesis example of Section 4.4.1.4. WOS-Synthesis.

FUNCTIONS OF THE INVENTORY

0
3
5
10
12
15
17
34
45
48
51
57
60
68
75
80
85
89
90
99
101
102
153
165
170
175

187
195
204
207
221
240
243
245
255
95
63

119

SYSTEM OF BOOLEAN EQUATIONS:

71000 701000 444444444 710 707 171 222222222

900000000

SOLUTIONS WITH LOWEST WEIGHT: W= 8

' SOLUTIONS NUMBER'

1

IDENTIFIER OF INPUTS

221

10
0
'SOLUTIONS NUMBER'
2
IDENTIFIER OF INPUTS
245
34
0
'SOLUTIONS NUMBER'
3
IDENTIFIER OF INPUTS
63
85
255

C. Output corresponding to synthesis using 3-input NAND.

FUNCTIONS OF THE INVENTORY

0
15
51
63
85
95
119
127
170
175
187
191
204
207
221
223
240
243
245
247
248
249
250
251
252
253
254
255

SYSTEM OF BOOLEAN EQUATIONS:

11000 70000 44444444 70 700 222222222
90000000

SOLUTIONS WITH LOWEST WEIGHT: W= 5

'SOLUTIONS NUMBER'

1

IDENTIFIER OF INPUTS

255

240

119

'SOLUTIONS NUMBER'

2

IDENTIFIER OF INPUTS

240

255

119

'SOLUTIONS NUMBER'

3

IDENTIFIER OF INPUTS

255

240

127

'SOLUTIONS NUMBER'

4

IDENTIFIER OF INPUTS

240

255

127

'SOLUTIONS NUMBER'
5
IDENTIFIER OF INPUTS
255
119
240
'SOLUTIONS NUMBER'
6
IDENTIFIER OF INPUTS
255
127
240
'SOLUTIONS NUMBER'
7
IDENTIFIER OF INPUTS
240
119
255
'SOLUTIONS NUMBER'
8
IDENTIFIER OF INPUTS
240
127
255
'SOLUTIONS NUMBER'
9
IDENTIFIER OF INPUTS
119
255
240
'SOLUTIONS NUMBER'
10
IDENTIFIER OF INPUTS
127
255
240
'SOLUTIONS NUMBER'
11
IDENTIFIER OF INPUTS
119
240
255
'SOLUTIONS NUMBER'
12
IDENTIFIER OF INPUTS
127
240
255

~~PRECEDING~~ PAGE BLANK NOT FILMED.

APPENDIX IV
BOOLEAN ANALYZER SIMULATOR

A. Program Listing

```

C
C****BAS IS A FORTRAN IV PROGRAM SIMULATING THE UCLA BOOLEAN ANALYZER
C****PROGRAMMER: MIGUEL A. MARIN, DEPART. OF ENGINEERING, U.C.L.A.
C
INTEG ER LOU(500,22),TC(22),LIT(22)
INTEG ER RAD(10000),BIT(32),TEMP,A,R
COMMON TC,LOU/STOR/RAD,BIT
READ (5,1000) (LIT(K),K=1,22)
DO 2 J=1,32
2 BIT(J)=2**(J-1)
1 READ (5,1001) MODE,N,M,NT
WRITE (6,808)
WRITE (6,808)
DO 102 I=1,10000
102 RAD(I)=0
808 FORMAT (1H0,'*****')
NTFF=NT
IF(NTF.LE.100) GO TO 4
NT=100
4 MODE = MODE + 1
GO TO (999,100,200),MODE
C
C****MODE I IN TRIADIC OPERATION
C****CANCELLATION OF NON-IMPLICANTS.
C
100 N1=3**N
WRITE (6,1022)

```



```

DO 5 I=1,NT
READ (5,1020) (LOU(I,K),K=1,22)
WRITE (6,1023) (LOU(I,K),K=1,22)
5 WRITE (6,1004) (LIT(K3),K3=1,22)
WRITE (6,808)
DO 110 I=1,N1
L=I-1
CALL BASE (L)
DO 108 K=1,NT
DO 104 J=1,22
TEMP=LOU(K,J)+TC(J)
IF(TEMP-3) 104,108,104
104 CONTINUE
R=1
L=I-1
CALL TRACK (L,A,R)
GOTO 110
108 CONTINUE
110 CONTINUE
NTF=NTF-100
IF(NTF.LE.0) GO TO 113
IF(NTF-100) 112,111,111
111 NT=100
GO TO 100
112 NT=NTF
GO TO 100

```

```

C
C*****MODE II
C*****PRIME IMPLICANT SELECTION
C

```

```

113 K=1
DO 114 K3=1,22
TC(K3)=0
114 LOU(K,K3)=2
K=0
KT=0
I=1
115 R=0
L=I-1
CALL MASK (A,K)
IF(A.NE.0) GOTO 128
CALL TRACK (L,A,R)
IF(A.NE.0) GO TO 128
120 L=I-1
CALL BASE (L)
121 K=K+1
DO 122 J=1,22
122 LOU(K,J)=TC(J)
IF(K.GT.100) GOTO131
123 L=I
CALL BASE (L)
CALL MASK (A,K)

```

```

124 I=I+1
128 IF(A) 129,130,129
129 I=I+1
    L=I-1
    CALL BASE (L)
130 IF(I.GT.N1) GO TO 145
    GO TO 115
131 L=L+1
    CALL BASE (L)
    CALL MASK (A,K)
    IF(A) 139,140,139
139 R=1
    CALL TRACK (L,A,R)
140 IF(L.LT.N1) GOTO131
    DO 142 IP=1,K
142 WRITE (9) (LOU(IP,IR),IR=1,22)
    KT=KT+K
    K=1
    DO 143 IS=1,22
143 LOU(K,IS)=2
    K=0
    GO TO 123
145 IF(KT.GT.100) GO TO 160
    WRITE (6,1006)
    DO 150 K1=1,K
    WRITE (6,1003) (LOU(K1,I),I=1,22)
150 WRITE (6,1004) (LIT(I),I=1,22)
    GO TO 1
160 KT=KT+K
    REWIND 9
    K=100
    WRITE (6,1006)
162 DO 163 IP=1,K
163 READ (9) (LOU(IP,IR),IR=1,22)
    DO 170 K1=1,K
    WRITE (6,1003) (LOU(K1,I),I=1,22)
170 WRITE (6,1004) (LIT(I),I=1,22)
    KT=KT-K
    IF(KT.GT.100) GOTO162
    K=KT
    IF(KT) 1,162,162
1000 FORMAT(22A1)
1001 FORMAT (4I3)
1003 FORMAT(1H0,22I1)
1004 FORMAT(1X,22A1)
1006 FORMAT (1H0,'LIST OF PRIME IMPLICANTS'/)
1020 FORMAT (22I1)
1022 FORMAT (1H1,'TERMS OF Y=0')
1023 FORMAT (1H0,22I1)

```

```

1024 FORMAT (1H0,'TERMS OF SYSTEM OF BOOLEAN EQUATIONS: Y=0')
C
C****MODE I IN BINARY OPERATION
C****SYSTEM-OF-BOOLEAN-EQUATIONS DISCRIMINANT DETERMINATION
C
200 WRITE (6,1024)
WRITE(6,1800) N,M
1800 FORMAT(1H0,'KNOWN VARIABLES=',I4,3X,'UNKNOWN VARIABLES=',I4)
DO 205 I=1,NT
READ (5,1020) (LOU(I,K),K=1,22)
WRITE (6,1023) (LOU(I,K),K=1,22)
205 WRITE (6,1004) (LIT(K),K=1,22)
WRITE (6,808)
N1=2** (N+M)
R=1
DO 220 I=1,N1
L=I-1
CALL BINARY (L)
DO 210 J=1,NT
DO 208 K=1,22
TEMP=TC(K)+LOU(J,K)
IF(TEMP-3) 208,210,208
208 CONTINUE
L=I-1
CALL TRACK (L,A,R)
GOTO 220
210 CONTINUE
220 CONTINUE
NTF=NTF-100
IF(NTF.LE.0) GO TO 215
IF(NTF-100) 212,211,211
211 NT=100
GO TO 200
212 NT=NTF
GO TO 200
215 WRITE (6,858)
858 FORMAT (1H0,'DISCRIMINANT')
CALL DISCRI (N,M)
GOTO 1
999 STOP
END

```

SUBROUTINE BASE (L)

C
C*****BASE THREE CONVERSION OF IDENTIFIER I .
C

INTEGER LOU(500,22),TC(22)
COMMON TC,LOU
DO 320 KL=1,22
TC(KL)=MOD(L,3)
320 L=L/3
RETURN
END

SUBROUTINE BINARY (L)

C
C*****BASE TWO CONVERSION OF IDENTIFIER L .
C

INTEGER LOU(500,22),TC(22)
INTEGER TEMP
COMMON TC,LOU
DO 520 K1=1,22
TEMP=MOD(L,2)
IF(TEMP) 510,512,510
510 TC(K1)=TEMP
GO TO 514
512 TC(K1)=2
514 L=L/2
520 CONTINUE
RETURN
END

SUBROUTINE MASK (A,K)

C
C*****CANCELLATION OF NON-PRIME-IMPLICANTS
C

INTEGER LOU(500,22),TC(22)
INTEGER TEMP,A
COMMON TC,LOU
DO 425 K2=1,K
DO 420 L=1,22
IF(LOU(K2,L)) 421,420,421
421 IF(LOU(K2,L)-TC(L)) 422,420,422
422 A=0
GO TO 425
420 CONTINUE
A=1
RETURN
425 CONTINUE
RETURN
END

```

SUBROUTINE TRACK (L,A,R)
C*****TRACK STORES CANCELLED TERM L IN RAD WHEN R=1
C*****TRACK READS OUT FROM RAD NON-CANCELLED TERM L WHEN R=0
C
INTEGER RAD (10000),BIT(32),WORD,POINT,A,R
COMMON/STOR/RAD,BIT
WORD=1+L/32
POINT=MOD(L,32)+1
IF (R) 600,700,600
600 RAD(WORD)=IOR(BIT(POINT),RAD(WORD))
RETURN
700 A=IAND(BIT(POINT),RAD(WORD))
RETURN
END

SUBROUTINE DISCRI (N,M)
C*****OUTPUT DISCRIMINANTE OF SYSTEM
C
INTEGER RAD(10000),B(32),WORD,ARRAY(8)
COMMON/STOR/RAD,BIT
M3=2**M
DO 820 I=1,M3
WORD=RAD(I)
CALL BIN (WORD,ARRAY)
820 WRITE (6,1100) ARRAY
1100 FORMAT (1X,8A4)
RETURN
END

```

1	00000001	R1	DEF	BIN
2	00000002	R2	EQU	1
3	00000003	R3	EQU	2
4	00000004	R4	EQU	3
5	00000005	K	EQU	4
6	00000006	J	EQU	5
7	00000007	I	EQU	6
8	00000000	ZEROS	EQU	7
9	FOFOFOFO	A	DATA	'0000'
10	00000001	A	DATA	X'00000001'
11	00000002		RES	16
12	00000012		BOUND	8
13	00000012		DATA	SREG-1
14	00000013		GEN, 1, 15, 1, 15	1, 16, 1, 0
15	00000014		RES	1
16	00000015		LCI	0
17	00000016		PSM, 0	SPD
18	00000017		BSO	ERROR
19	00000018		LI, R1	1
20	00000019		LW, R2	*13, R1
21	0000001A		LW, R2	*R2
22	0000001B		LI, R1	2
23	0000001C		LW, R1	*13, R1
24	00001D		AI, R1	-1
25	00001E		STW, R1	ADDRESS
26	00001F		LI, K	0
27	000020		LI, J	8
28	000021	HERE	LW, R3	ZEROS
29	000022		LI, I	4
30	000023	THERE	LW, R4	BITMASK
31	000024		SLS, R4	0, K
32	000025		AND, R4	R2

33	00026	22100004	A	LI, R1	4
34	00027	38100007	A	SW, R1	I
35	00028	25100003	A	SLS, R1	3
36	00029	38100005	A	SW, R1	K
37	0002A	25420000	A	SLS, R4	O, R1
38	0002B	49300004	A	OR, R3	R4
39	0002C	20500001	A	AI, K	1
40	0002D	64700023	A	BDR, I	THERE
41	0002E	B53C0014	A	STW, R3	*ADDRESS, J
42	0002F	64600021	A	BDR, J	HERE
43	00030	02200000	A	LCI	0
44	00031	0A000012	A	PLM, 0	SPD
45	00032	20D00003	A	AI, 13	3
46	00033	E800000D	A	B	*13
47	00034	04200000	F	CALL1, 2	ERFPT
48	00035	E800000D	A	B	*13
49	00036	02000000	A	GEN, 8, 24	X'02', 0
50	00037	80000000	A	GEN, 1, 31	1, 0
51	00038	00000000	N	DATA	ERMSG
52	00039	15E2E3C1	A	TEXTC	'STACK OVERFLOW IN BIN'
	0003A	C3D240D6	A		
	0003B	E5C5D9C6	A		
	0003C	D3D6E640	A		
	0003D	C9D540C2	A		
	0003E	C9D54040	A		
					END

1					DEF	IAND
2	00000				BOUND	4
3	00000	35700000	F	IAND	STW,7	SAVE1
4	00001	35600000	F		STW,6	SAVE2
5	00002	3270000D	A		LW,7	13
6	00003	20700003	A		AI,7	3
7	00004	35700000	F		STW,7	RET
8	00005	20D00001	A		AI,13	1
9	00006	B270000D	A		LW,7	*13
10	00007	B2300007	A		LW,3	*7
11	00008	20D00001	A		AI,13	1
12	00009	B270000D	A		LW,7	*13
13	0000A	B2600007	A		LW,6	*7
14	0000B	4B300006	A		AND,3	6
15	0000C	32700000	F		LW,7	SAVE1
16	0000D	32600000	F		LW,6	SAVE2
17	0000E	E8000000	F		B	*RET
18	0000F			SAVE1	RES	1
19	00010			SAVE2	RES	1
20	00011			RET	RES	1
21					END	

1					DEF	IOR
2	00000				BOUND	4
3	00000	35700000	F	IOR	STW,7	SAVE1
4	00001	35600000	F		STW,6	SAVE2
5	00002	3270000D	A		LW,7	13
6	00003	20700003	A		AI,7	3
7	00004	35700000	F		STW,7	RET
8	00005	20D00001	A		AI,13	1
9	00006	B270000D	A		LW,7	*13
10	00007	B2300007	A		LW,3	*7
11	00008	20D00001	A		AI,13	1
12	00009	B270000D	A		LW,7	*13
13	0000A	B2600007	A		LW,6	*7
14	0000B	49300006	A		OR,3	6
15	0000C	32700000	F		LW,7	SAVE1
16	0000D	32600000	F		LW,6	SAVE2
17	0000E	E8000000	F		B	*RET
18	0000F			SAVE1	RES	1
19	00010			SAVE2	RES	1
20	00011			RET	RES	1
21					END	

B. System of Boolean Equations Discriminant Determination.

Output corresponding to Equation (10), Section 1.1.3.

```
*****
*****
TERMS OF SYSTEM OF BOOLEAN EQUATIONS: Y=0
KNOWN VARIABLES= 2   UNKNOWN VARIABLES= 2
020100000000000000000000
ABCDEFGHJKLMNOPQRSTUWVZ
022000000000000000000000
ABCDEFGHJKLMNOPQRSTUWVZ
102200000000000000000000
ABCDEFGHJKLMNOPQRSTUWVZ
211000000000000000000000
ABCDEFGHJKLMNOPQRSTUWVZ
*****
DISCRIMINANT
000000000000000000111001101001011
0000000000000000000000000000000000
0000000000000000000000000000000000
0000000000000000000000000000000000
```

C. Prime Implicant Calculation. Output corresponding to Tables II and III (Section 3.1.).

```
*****
*****
TERMS OF Y=0
021000000000000000000000
ABCDEFGHJKLMNOPQRSTUWVZ
220000000000000000000000
ABCDEFGHJKLMNOPQRSTUWVZ
*****
LIST OF PRIME IMPLICANTS
010000000000000000000000
ABCDEFGHJKLMNOPQRSTUWVZ
102000000000000000000000
ABCDEFGHJKLMNOPQRSTUWVZ
*****
*****
```

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Department of Engineering University of California Los Angeles, Calif. 90024		2a. REPORT SECURITY CLASSIFICATION	
		2b. GROUP	
3. REPORT TITLE Investigation of the Field of Problems for the Boolean Analyzer			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates)			
5. AUTHOR(S) (Last name, first name, initial) Marin, Miguel A.			
6. REPORT DATE June 1968	7a. TOTAL NO. OF PAGES 173	7b. NO. OF REFS 40	
8a. CONTRACT OR GRANT NO. Nonr 233(52) SD-184,	9a. ORIGINATOR'S REPORT NUMBER(S) 68-28		
b. PROJECT NO. AT(11-1) Gen 10, Proj. 14	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)		
c. NGRO5-007-201			
d.			
10. AVAILABILITY/LIMITATION NOTICES Distribution of this Document is Unlimited			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY Office of Naval Research, AEC, Advanced Research Project Agency	
13. ABSTRACT Recently, a new processor, called Boolean Analyzer (BA), has been proposed by A. Svoboda. This unit operating as a part of an automatic computer is based on the idea of processing many terms of Boolean Algebra in parallel. One of its operational capabilities is the solution of large systems of Boolean equations in a reasonable time (if we compare it to standard procedures). In order to promote the use of the Boolean Analyzer in its two modes of operation we developed some applications to logic problems. In this report we propose to show how the Boolean Analyzer should be integrated with a general purpose computer (or with a variable structure computer) to solve efficiently: 1) the <u>Three-level AND-NOT</u> synthesis problem of logic networks using only <u>True</u> inputs (Synthesis of TANT networks), and 2) the general problem of synthesis of logic networks using a restricted inventory of integrated circuit modules. Our task includes reformulation of those problems into Boolean equations to prove their solubility by a Boolean Analyzer integrated with another system.			

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Boolean Analyzer TANT Networks Logic Design Synthesis of Logic Nets Boolean Equations Computer Design Automatic Logic Design Prime Implicants Determination Variable Structure Computer						

INSTRUCTIONS

1. **ORIGINATING ACTIVITY:** Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization (*corporate author*) issuing the report.
- 2a. **REPORT SECURITY CLASSIFICATION:** Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.
- 2b. **GROUP:** Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.
3. **REPORT TITLE:** Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.
4. **DESCRIPTIVE NOTES:** If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.
5. **AUTHOR(S):** Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.
6. **REPORT DATE:** Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.
- 7a. **TOTAL NUMBER OF PAGES:** The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.
- 7b. **NUMBER OF REFERENCES:** Enter the total number of references cited in the report.
- 8a. **CONTRACT OR GRANT NUMBER:** If appropriate, enter the applicable number of the contract or grant under which the report was written.
- 8b, 8c, & 8d. **PROJECT NUMBER:** Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.
- 9a. **ORIGINATOR'S REPORT NUMBER(S):** Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.
- 9b. **OTHER REPORT NUMBER(S):** If the report has been assigned any other report numbers (*either by the originator or by the sponsor*), also enter this number(s).
10. **AVAILABILITY/LIMITATION NOTICES:** Enter any limitations on further dissemination of the report, other than those

imposed by security classification, using standard statements such as:

- (1) "Qualified requesters may obtain copies of this report from DDC."
- (2) "Foreign announcement and dissemination of this report by DDC is not authorized."
- (3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."
- (4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."
- (5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. **SUPPLEMENTARY NOTES:** Use for additional explanatory notes.
12. **SPONSORING MILITARY ACTIVITY:** Enter the name of the departmental project office or laboratory sponsoring (*paying for*) the research and development. Include address.
13. **ABSTRACT:** Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as (TS), (S), (C), or (U).

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. **KEY WORDS:** Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, roles, and weights is optional.

DISTRIBUTION LIST

TECHNICAL LIBRARY
DIRECTOR DEFENSE RES. & ENG.
ROOM 3C-128, THE PENTAGON
WASHINGTON, D.C. 20301

DEFENSE DOCUMENTATION CENTER
CAMERON STATION
ALEXANDRIA, VIRGINIA 22314

20 COPIES

CHIEF OF NAVAL RESEARCH
DEPARTMENT OF THE NAVY
WASHINGTON, D.C. 20360
ATTN CODE 437, INFORMATION SYSTEMS BRANCH

2 COPIES

DIRECTOR, NAVAL RESEARCH LABORATORY
TECHNICAL INFORMATION OFFICER/CODE 2000
WASHINGTON, D.C. 20390

6 COPIES

COMMANDING OFFICER, OFFICE OF NAVAL RESEARCH
FLEET POST OFFICE BOX 39
NEW YORK, NEW YORK 09510

10 COPIES

COMMANDING OFFICER
SCIENTIFIC DEPT.
ONR BRANCH OFFICE
207 WEST 24TH STREET
NEW YORK, NEW YORK 10011

OFFICE OF NAVAL RESEARCH BRANCH OFFICE
495 SUMMER STREET
BOSTON, MASSACHUSETTS 02110

U.S. ATTN. EVA LIBERMAN, LIBRARIAN
TECHNICAL LIBRARY
NAVAL ORDNANCE LABORATORY
WHITE OAK
SILVER SPRING, MARYLAND 20910

DAVID TAYLOR MODEL BASIN
CODE 042 TECHNICAL LIBRARY
WASHINGTON, D.C. 20007

COMMANDING OFFICER & DIRECTOR
U.S. NAVY ELECTRONICS LABORATORY/LIBRARY
SAN DIEGO, CALIFORNIA 92152

DR. W. DALE COMPTON
COORDINATED SCIENCE LABORATORY
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS 61801

AIR FORCE CAMBRIDGE RESEARCH LABS
LAURENCE C. HANSCOM FIELD
BEDFORD, MASSACHUSETTS
ATTN RESEARCH LIBRARY, CRMXL-R 01731

DISTRIBUTION LIST (Continued)

MR. GENE H. GLEISSNER
APPLIED MATH. LAB./CODE 800
DAVID TAYLOR MODEL BASIN
WASHINGTON, D.C. 20007

U.S. NAVY ELECTRONICS LABORATORY
JOHN SLAUGHTER
CODE 3350E
SAN DIEGO, CALIFORNIA 92152

MRS. ANNA K. SMILON, PUBL. EDITOR
CENTER FOR COMPUTER SCIENCES & TECH.
NATIONAL BUREAU OF STANDARDS
WASHINGTON, D.C. 20234

DR. NILS NILSSON
ARTIFICIAL INTELLIGENCE GROUP
STANFORD RESEARCH INSTITUTE
MENLO PARK, CALIFORNIA 94025

GEORGE C. FRANCIS
COMPUTING LAB. BRL
ABERDEEN PROVING GROUND, MARYLAND 21005

OFFICE OF NAVAL RESEARCH
BRANCH OFFICE CHICAGO
219 S. MICHIGAN AVENUE
CHICAGO, ILLINOIS 60604

DR. ARTHUR R. LAUFER
COMMANDING OFFICER
ONR BRANCH OFFICE
SCIENTIFIC DEPT.
1030 E. GREEN STREET
PASADENA, CALIFORNIA 91101

AUTONOMICS DIVISION
NATIONAL PHYSICAL LABORATORY
TEDDINGTON, MIDDLESEX
ENGLAND
ATTN DR. A.M. UTTLEY, SUPERINTENDENT

DIRECTOR NATIONAL SECURITY AGENCY
FORT GEORGE G. MEADE, MARYLAND 20755
ATTN C3/TDL

LINCOLN LABORATORY
LIBRARY A-082
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
LEXINGTON, MASSACHUSETTS 02173

COMPUTER COMMAND AND CONTROL CORP.
SUITE 1315
1750 PENNSYLVANIA AVENUE N W
WASHINGTON, D.C. 20006
ATTN MR. WILLIAM C. MANN

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF ELECTRICAL ENGINEERING
CAMBRIDGE, MASSACHUSETTS 02139
ATTN PROFESSOR J.F. REINTJES

PROJECT MAC-MIT
545 TECHNOLOGY SQUARE
CAMBRIDGE, MASSACHUSETTS 02139
ATTN PROF. R.M. FANO